

ChibiOS/RT

3.1.5

Reference Manual

Sun Jan 1 2017 18:09:20

Contents

1 ChibiOS/RT	1
1.1 Copyright	1
1.2 Introduction	1
1.3 Related Documents	1
2 Kernel Concepts	3
2.1 Naming Conventions	3
2.2 API Name Suffixes	3
2.3 Interrupt Classes	4
2.4 System States	4
2.5 Scheduling	6
2.6 Thread States	7
2.7 Priority Levels	7
2.8 Thread Working Area	7
3 Testing Strategy	9
3.1 Threads and Scheduler test	9
3.1.1 Ready List functionality #1	10
3.1.2 Ready List functionality #2	10
3.1.3 Threads priority change test	10
3.1.4 Threads delays test	10
3.2 Dynamic APIs test	11
3.2.1 Threads creation from Memory Heap	11
3.2.2 Threads creation from Memory Pool	11
3.2.3 Registry and References test	11
3.3 Messages test	12
3.3.1 Messages Server loop	12
3.4 Semaphores test	12
3.4.1 Enqueuing test	13
3.4.2 Timeout test	13
3.4.3 Atomic signal-wait test	13
3.4.4 Binary Wait and Signal	13

3.5	Mutexes test	13
3.5.1	Priority enqueueing test	14
3.5.2	Priority inheritance, simple case	14
3.5.3	Priority inheritance, complex case	15
3.5.4	Priority return verification	15
3.5.5	Mutex status	15
3.5.6	Condition Variable signal test	15
3.5.7	Condition Variable broadcast test	16
3.5.8	Condition Variable priority boost test	16
3.6	Events test	16
3.6.1	Events registration and dispatch	17
3.6.2	Events wait and broadcast	17
3.6.3	Events timeout	17
3.7	Mailboxes test	17
3.7.1	Queuing and timeouts	18
3.8	I/O Queues test	18
3.8.1	Input Queues functionality and APIs	19
3.8.2	Output Queues functionality and APIs	19
3.9	Memory Heap test	19
3.9.1	Allocation and fragmentation test	19
3.10	Memory Pools test	19
3.10.1	Allocation and enqueueing test	20
3.11	System test	20
3.11.1	Critical zones check	21
3.11.2	Interrupts handling	21
3.11.3	System integrity check	21
3.12	Kernel Benchmarks	21
3.12.1	Messages performance #1	22
3.12.2	Messages performance #2	22
3.12.3	Messages performance #3	22
3.12.4	Context Switch performance	22
3.12.5	Threads performance, full cycle	22
3.12.6	Threads performance, create/exit only	22
3.12.7	Mass reschedule performance	23
3.12.8	I/O Round-Robin voluntary reschedule	23
3.12.9	I/O Queues throughput	23
3.12.10	Virtual Timers set/reset performance	23
3.12.11	Semaphores wait/signal performance	23
3.12.12	Mutexes lock/unlock performance	23
3.12.13	RAM Footprint	24

4	Deprecated List	25
5	Module Index	27
5.1	Modules	27
6	Hierarchical Index	29
6.1	Class Hierarchy	29
7	Data Structure Index	31
7.1	Data Structures	31
8	File Index	33
8.1	File List	33
9	Module Documentation	37
9.1	RT Kernel	37
9.1.1	Detailed Description	37
9.2	Version Numbers and Identification	38
9.2.1	Detailed Description	38
9.2.2	Macro Definition Documentation	38
9.2.2.1	_CHIBIOS_RT_	38
9.2.2.2	CH_KERNEL_STABLE	38
9.2.2.3	CH_KERNEL_VERSION	38
9.2.2.4	CH_KERNEL_MAJOR	38
9.2.2.5	CH_KERNEL_MINOR	38
9.2.2.6	CH_KERNEL_PATCH	38
9.3	Configuration	39
9.3.1	Detailed Description	39
9.3.2	Macro Definition Documentation	41
9.3.2.1	CH_CFG_ST_RESOLUTION	41
9.3.2.2	CH_CFG_ST_FREQUENCY	41
9.3.2.3	CH_CFG_ST_TIMEDELTA	41
9.3.2.4	CH_CFG_TIME_QUANTUM	41
9.3.2.5	CH_CFG_MEMCORE_SIZE	42
9.3.2.6	CH_CFG_NO_IDLE_THREAD	42
9.3.2.7	CH_CFG_OPTIMIZE_SPEED	42
9.3.2.8	CH_CFG_USE_TM	42
9.3.2.9	CH_CFG_USE_REGISTRY	42
9.3.2.10	CH_CFG_USE_WAITEXIT	42
9.3.2.11	CH_CFG_USE_SEMAPHORES	43
9.3.2.12	CH_CFG_USE_SEMAPHORES_PRIORITY	43
9.3.2.13	CH_CFG_USE_MUTEXES	43

9.3.2.14 CH_CFG_USE_MUTEXES_RECURSIVE	43
9.3.2.15 CH_CFG_USE_CONDVARNS	43
9.3.2.16 CH_CFG_USE_CONDVARNS_TIMEOUT	44
9.3.2.17 CH_CFG_USE_EVENTS	44
9.3.2.18 CH_CFG_USE_EVENTS_TIMEOUT	44
9.3.2.19 CH_CFG_USE_MESSAGES	44
9.3.2.20 CH_CFG_USE_MESSAGES_PRIORITY	44
9.3.2.21 CH_CFG_USE_MAILBOXES	45
9.3.2.22 CH_CFG_USE_QUEUES	45
9.3.2.23 CH_CFG_USE_MEMCORE	45
9.3.2.24 CH_CFG_USE_HEAP	45
9.3.2.25 CH_CFG_USE_MEMPOOLS	45
9.3.2.26 CH_CFG_USE_DYNAMIC	46
9.3.2.27 CH_DBG_STATISTICS	46
9.3.2.28 CH_DBG_SYSTEM_STATE_CHECK	46
9.3.2.29 CH_DBG_ENABLE_CHECKS	46
9.3.2.30 CH_DBG_ENABLE_ASSERTS	46
9.3.2.31 CH_DBG_ENABLE_TRACE	47
9.3.2.32 CH_DBG_ENABLE_STACK_CHECK	47
9.3.2.33 CH_DBG_FILL_THREADS	47
9.3.2.34 CH_DBG_THREADS_PROFILING	47
9.3.2.35 CH_CFG_THREAD_EXTRA_FIELDS	47
9.3.2.36 CH_CFG_THREAD_INIT_HOOK	47
9.3.2.37 CH_CFG_THREAD_EXIT_HOOK	48
9.3.2.38 CH_CFG_CONTEXT_SWITCH_HOOK	48
9.3.2.39 CH_CFG_IDLE_ENTER_HOOK	48
9.3.2.40 CH_CFG_IDLE_LEAVE_HOOK	48
9.3.2.41 CH_CFG_IDLE_LOOP_HOOK	49
9.3.2.42 CH_CFG_SYSTEM_TICK_HOOK	49
9.3.2.43 CH_CFG_SYSTEM_HALT_HOOK	49
9.4 Kernel Types	50
9.4.1 Detailed Description	50
9.4.2 Macro Definition Documentation	50
9.4.2.1 FALSE	50
9.4.2.2 TRUE	50
9.4.2.3 ROMCONST	51
9.4.2.4 NOINLINE	51
9.4.2.5 PORT_THD_FUNCTION	51
9.4.2.6 PACKED_VAR	51
9.4.3 Typedef Documentation	51

9.4.3.1	<code>rtcnt_t</code>	51
9.4.3.2	<code>rttime_t</code>	51
9.4.3.3	<code>syssts_t</code>	51
9.4.3.4	<code>tmode_t</code>	51
9.4.3.5	<code>tstate_t</code>	51
9.4.3.6	<code>trefs_t</code>	51
9.4.3.7	<code>tslices_t</code>	52
9.4.3.8	<code>tprio_t</code>	52
9.4.3.9	<code>msg_t</code>	52
9.4.3.10	<code>eventid_t</code>	52
9.4.3.11	<code>eventmask_t</code>	52
9.4.3.12	<code>eventflags_t</code>	52
9.4.3.13	<code>cnt_t</code>	52
9.4.3.14	<code>ucnt_t</code>	52
9.5	Base Kernel Services	53
9.5.1	Detailed Description	53
9.6	System Management	54
9.6.1	Detailed Description	54
9.6.2	Macro Definition Documentation	56
9.6.2.1	<code>CH_IRQ_IS_VALID_PRIORITY</code>	56
9.6.2.2	<code>CH_IRQ_IS_VALID_KERNEL_PRIORITY</code>	56
9.6.2.3	<code>CH_IRQ_PROLOGUE</code>	56
9.6.2.4	<code>CH_IRQ_EPILOGUE</code>	57
9.6.2.5	<code>CH_IRQ_HANDLER</code>	57
9.6.2.6	<code>CH_FAST_IRQ_HANDLER</code>	57
9.6.2.7	<code>S2RTC</code>	58
9.6.2.8	<code>MS2RTC</code>	58
9.6.2.9	<code>US2RTC</code>	58
9.6.2.10	<code>RTC2S</code>	59
9.6.2.11	<code>RTC2MS</code>	59
9.6.2.12	<code>RTC2US</code>	60
9.6.2.13	<code>chSysGetRealtimeCounterX</code>	60
9.6.2.14	<code>chSysSwitch</code>	60
9.6.3	Function Documentation	61
9.6.3.1	<code>_idle_thread</code>	61
9.6.3.2	<code>chSysInit</code>	61
9.6.3.3	<code>chSysHalt</code>	62
9.6.3.4	<code>chSysIntegrityCheck</code>	63
9.6.3.5	<code>chSysTimerHandler</code>	64
9.6.3.6	<code>chSysGetStatusAndLockX</code>	64

9.6.3.7	chSysRestoreStatusX	65
9.6.3.8	chSysIsCounterWithinX	66
9.6.3.9	chSysPolledDelayX	67
9.6.3.10	chSysDisable	67
9.6.3.11	chSysSuspend	68
9.6.3.12	chSysEnable	68
9.6.3.13	chSysLock	69
9.6.3.14	chSysUnlock	69
9.6.3.15	chSysLockFromISR	70
9.6.3.16	chSysUnlockFromISR	70
9.6.3.17	chSysUnconditionalLock	71
9.6.3.18	chSysUnconditionalUnlock	72
9.6.3.19	chSysGetIdleThreadX	72
9.7	Scheduler	73
9.7.1	Detailed Description	73
9.7.2	Macro Definition Documentation	77
9.7.2.1	MSG_OK	77
9.7.2.2	MSG_TIMEOUT	77
9.7.2.3	MSG_RESET	77
9.7.2.4	NOPRIO	77
9.7.2.5	IDLEPRIO	77
9.7.2.6	LOWPRIO	77
9.7.2.7	NORMALPRIO	77
9.7.2.8	HIGHPRIO	77
9.7.2.9	ABSPRIO	77
9.7.2.10	CH_STATE_READY	78
9.7.2.11	CH_STATE_CURRENT	78
9.7.2.12	CH_STATE_WTSTART	78
9.7.2.13	CH_STATE_SUSPENDED	78
9.7.2.14	CH_STATE_QUEUED	78
9.7.2.15	CH_STATE_WTSEM	78
9.7.2.16	CH_STATE_WTMTX	78
9.7.2.17	CH_STATE_WTCOND	78
9.7.2.18	CH_STATE_SLEEPING	78
9.7.2.19	CH_STATE_WTEXIT	78
9.7.2.20	CH_STATE_WTOREVT	78
9.7.2.21	CH_STATE_WTANDEV	78
9.7.2.22	CH_STATE_SNDMSGQ	79
9.7.2.23	CH_STATE_SNDMSG	79
9.7.2.24	CH_STATE_WTMSG	79

9.7.2.25	CH_STATE_FINAL	79
9.7.2.26	CH_STATE_NAMES	79
9.7.2.27	CH_FLAG_MODE_MASK	79
9.7.2.28	CH_FLAG_MODE_STATIC	79
9.7.2.29	CH_FLAG_MODE_HEAP	79
9.7.2.30	CH_FLAG_MODE_MPOOL	79
9.7.2.31	CH_FLAG_TERMINATE	79
9.7.2.32	THD_ALIGN_STACK_SIZE	79
9.7.2.33	THD_WORKING_AREA_SIZE	80
9.7.2.34	THD_WORKING_AREA	80
9.7.2.35	THD_FUNCTION	80
9.7.2.36	firstprio	80
9.7.2.37	currp	81
9.7.2.38	setcurrp	81
9.7.3	Typedef Documentation	81
9.7.3.1	systime_t	81
9.7.3.2	thread_t	81
9.7.3.3	thread_reference_t	81
9.7.3.4	threads_list_t	81
9.7.3.5	threads_queue_t	81
9.7.3.6	ready_list_t	81
9.7.3.7	vtfunc_t	81
9.7.3.8	virtual_timer_t	82
9.7.3.9	virtual_timers_list_t	82
9.7.3.10	system_debug_t	82
9.7.3.11	ch_system_t	82
9.7.4	Function Documentation	82
9.7.4.1	_scheduler_init	82
9.7.4.2	queue_prio_insert	82
9.7.4.3	queue_insert	83
9.7.4.4	queue_fifo_remove	83
9.7.4.5	queue_lifo_remove	83
9.7.4.6	queue_dequeue	84
9.7.4.7	list_insert	84
9.7.4.8	list_remove	84
9.7.4.9	chSchReadyI	84
9.7.4.10	chSchGoSleepS	85
9.7.4.11	chSchGoSleepTimeoutS	86
9.7.4.12	chSchWakeUpS	87
9.7.4.13	chSchRescheduleS	88

9.7.4.14	chSchIsPreemptionRequired	88
9.7.4.15	chSchDoRescheduleBehind	88
9.7.4.16	chSchDoRescheduleAhead	89
9.7.4.17	chSchDoReschedule	89
9.7.4.18	list_init	90
9.7.4.19	list_isempty	90
9.7.4.20	list_notempty	90
9.7.4.21	queue_init	91
9.7.4.22	queue_isempty	92
9.7.4.23	queue_notempty	92
9.7.4.24	chSchIsRescRequired!	92
9.7.4.25	chSchCanYieldS	93
9.7.4.26	chSchDoYieldS	93
9.7.4.27	chSchPreemption	94
9.7.5	Variable Documentation	94
9.7.5.1	ch	94
9.8	Threads	95
9.8.1	Detailed Description	95
9.8.2	Macro Definition Documentation	97
9.8.2.1	_THREADS_QUEUE_DATA	97
9.8.2.2	_THREADS_QUEUE_DECL	97
9.8.2.3	chThdSleepSeconds	97
9.8.2.4	chThdSleepMilliseconds	98
9.8.2.5	chThdSleepMicroseconds	98
9.8.3	Typedef Documentation	98
9.8.3.1	tfunc_t	98
9.8.4	Function Documentation	98
9.8.4.1	_thread_init	98
9.8.4.2	_thread_memfill	99
9.8.4.3	chThdCreate!	99
9.8.4.4	chThdCreateStatic	100
9.8.4.5	chThdStart	101
9.8.4.6	chThdSetPriority	102
9.8.4.7	chThdTerminate	103
9.8.4.8	chThdSleep	104
9.8.4.9	chThdSleepUntil	105
9.8.4.10	chThdSleepUntilWindowed	106
9.8.4.11	chThdYield	107
9.8.4.12	chThdExit	108
9.8.4.13	chThdExitS	109

9.8.4.14	chThdWait	110
9.8.4.15	chThdSuspendS	111
9.8.4.16	chThdSuspendTimeoutS	112
9.8.4.17	chThdResumel	113
9.8.4.18	chThdResumeS	113
9.8.4.19	chThdResume	114
9.8.4.20	chThdEnqueueTimeoutS	115
9.8.4.21	chThdDequeueNextl	116
9.8.4.22	chThdDequeueAlll	116
9.8.4.23	chThdGetSelfX	117
9.8.4.24	chThdGetPriorityX	117
9.8.4.25	chThdGetTicksX	118
9.8.4.26	chThdTerminatedX	118
9.8.4.27	chThdShouldTerminateX	118
9.8.4.28	chThdStartl	119
9.8.4.29	chThdSleepS	119
9.8.4.30	chThdQueueObjectInit	120
9.8.4.31	chThdQueueIsEmptyl	120
9.8.4.32	chThdDoDequeueNextl	121
9.9	Time and Virtual Timers	123
9.9.1	Detailed Description	123
9.9.2	Macro Definition Documentation	124
9.9.2.1	TIME_IMMEDIATE	124
9.9.2.2	TIME_INFINITE	124
9.9.2.3	S2ST	124
9.9.2.4	MS2ST	125
9.9.2.5	US2ST	125
9.9.2.6	ST2S	125
9.9.2.7	ST2MS	126
9.9.2.8	ST2US	126
9.9.3	Function Documentation	127
9.9.3.1	_vt_init	127
9.9.3.2	chVTDoSetl	127
9.9.3.3	chVTDoResetl	128
9.9.3.4	chVTOBJECTINIT	129
9.9.3.5	chVTGetSystemTimeX	129
9.9.3.6	chVTGetSystemTime	129
9.9.3.7	chVTTIMEELAPSED SINCE X	130
9.9.3.8	chVTIS TIME WITHIN X	130
9.9.3.9	chVTIS SYSTEM TIME WITHIN X	131

9.9.3.10 chVTIsSystemTimeWithin	132
9.9.3.11 chVTGetTimersStateI	132
9.9.3.12 chVTIsArmedI	133
9.9.3.13 chVTIsArmed	134
9.9.3.14 chVTResetI	134
9.9.3.15 chVTReset	135
9.9.3.16 chVTSetI	136
9.9.3.17 chVTSet	137
9.9.3.18 chVTDoTickI	138
9.10 Synchronization	140
9.10.1 Detailed Description	140
9.11 Counting Semaphores	141
9.11.1 Detailed Description	141
9.11.2 Macro Definition Documentation	142
9.11.2.1 _SEMAPHORE_DATA	142
9.11.2.2 SEMAPHORE_DECL	142
9.11.3 Typedef Documentation	143
9.11.3.1 semaphore_t	143
9.11.4 Function Documentation	143
9.11.4.1 chSemObjectInit	143
9.11.4.2 chSemReset	143
9.11.4.3 chSemResetI	144
9.11.4.4 chSemWait	145
9.11.4.5 chSemWaitS	146
9.11.4.6 chSemWaitTimeout	147
9.11.4.7 chSemWaitTimeoutS	148
9.11.4.8 chSemSignal	149
9.11.4.9 chSemSignall	150
9.11.4.10 chSemAddCounterI	151
9.11.4.11 chSemSignalWait	152
9.11.4.12 chSemFastWaitI	153
9.11.4.13 chSemFastSignall	154
9.11.4.14 chSemGetCounterI	154
9.12 Binary Semaphores	156
9.12.1 Detailed Description	156
9.12.2 Macro Definition Documentation	157
9.12.2.1 _BSEMAPHORE_DATA	157
9.12.2.2 BSEMAPHORE_DECL	157
9.12.3 Function Documentation	157
9.12.3.1 chBSemObjectInit	157

9.12.3.2 chBSemWait	158
9.12.3.3 chBSemWaitS	159
9.12.3.4 chBSemWaitTimeoutS	160
9.12.3.5 chBSemWaitTimeout	161
9.12.3.6 chBSemResetI	162
9.12.3.7 chBSemReset	163
9.12.3.8 chBSemSignall	163
9.12.3.9 chBSemSignal	164
9.12.3.10 chBSemGetStatel	165
9.13 Mutexes	167
9.13.1 Detailed Description	167
9.13.2 Macro Definition Documentation	168
9.13.2.1 _MUTEX_DATA	168
9.13.2.2 MUTEX_DECL	168
9.13.3 Typedef Documentation	169
9.13.3.1 mutex_t	169
9.13.4 Function Documentation	169
9.13.4.1 chMtxObjectInit	169
9.13.4.2 chMtxLock	169
9.13.4.3 chMtxLockS	170
9.13.4.4 chMtxTryLock	171
9.13.4.5 chMtxTryLockS	172
9.13.4.6 chMtxUnlock	173
9.13.4.7 chMtxUnlockS	174
9.13.4.8 chMtxUnlockAll	175
9.13.4.9 chMtxQueueNotEmptyS	176
9.13.4.10 chMtxGetNextMutexS	177
9.14 Condition Variables	178
9.14.1 Detailed Description	178
9.14.2 Macro Definition Documentation	179
9.14.2.1 _CONDVAR_DATA	179
9.14.2.2 CONDVAR_DECL	179
9.14.3 Typedef Documentation	179
9.14.3.1 condition_variable_t	179
9.14.4 Function Documentation	179
9.14.4.1 chCondObjectInit	179
9.14.4.2 chCondSignal	180
9.14.4.3 chCondSignall	180
9.14.4.4 chCondBroadcast	181
9.14.4.5 chCondBroadcastl	182

9.14.4.6 chCondWait	183
9.14.4.7 chCondWaitS	184
9.14.4.8 chCondWaitTimeout	185
9.14.4.9 chCondWaitTimeoutS	186
9.15 Event Flags	188
9.15.1 Detailed Description	188
9.15.2 Macro Definition Documentation	190
9.15.2.1 ALL_EVENTS	190
9.15.2.2 EVENT_MASK	190
9.15.2.3 _EVENTSOURCE_DATA	190
9.15.2.4 EVENTSOURCE_DECL	190
9.15.3 Typedef Documentation	190
9.15.3.1 event_source_t	190
9.15.3.2 evhandler_t	190
9.15.4 Function Documentation	190
9.15.4.1 chEvtRegisterMaskWithFlags	190
9.15.4.2 chEvtUnregister	191
9.15.4.3 chEvtGetAndClearEvents	192
9.15.4.4 chEvtAddEvents	193
9.15.4.5 chEvtBroadcastFlags!	194
9.15.4.6 chEvtGetAndClearFlags!	195
9.15.4.7 chEvtSignal	195
9.15.4.8 chEvtSignall	196
9.15.4.9 chEvtBroadcastFlags	197
9.15.4.10 chEvtGetAndClearFlags!	198
9.15.4.11 chEvtDispatch	198
9.15.4.12 chEvtWaitOne	199
9.15.4.13 chEvtWaitAny	200
9.15.4.14 chEvtWaitAll	201
9.15.4.15 chEvtWaitOneTimeout	202
9.15.4.16 chEvtWaitAnyTimeout	203
9.15.4.17 chEvtWaitAllTimeout	204
9.15.4.18 chEvtObjectInit	205
9.15.4.19 chEvtRegisterMask	206
9.15.4.20 chEvtRegister	206
9.15.4.21 chEvtIsListening!	207
9.15.4.22 chEvtBroadcast	207
9.15.4.23 chEvtBroadcast!	208
9.16 Synchronous Messages	209
9.16.1 Detailed Description	209

9.16.2 Function Documentation	209
9.16.2.1 chMsgSend	209
9.16.2.2 chMsgWait	210
9.16.2.3 chMsgRelease	211
9.16.2.4 chMsgIsPendingl	212
9.16.2.5 chMsgGet	213
9.16.2.6 chMsgReleaseS	213
9.17 Mailboxes	215
9.17.1 Detailed Description	215
9.17.2 Macro Definition Documentation	216
9.17.2.1 _MAILBOX_DATA	216
9.17.2.2 MAILBOX_DECL	216
9.17.3 Function Documentation	217
9.17.3.1 chMBOBJECTInit	217
9.17.3.2 chMBReset	217
9.17.3.3 chMBResetl	218
9.17.3.4 chMBPost	219
9.17.3.5 chMBPostS	220
9.17.3.6 chMBPostl	221
9.17.3.7 chMBPostAhead	222
9.17.3.8 chMBPostAheadS	223
9.17.3.9 chMBPostAheadl	224
9.17.3.10 chMBFetch	225
9.17.3.11 chMBFetchS	226
9.17.3.12 chMBFetchl	227
9.17.3.13 chMBGetSizel	228
9.17.3.14 chMBGetFreeCountl	228
9.17.3.15 chMBGetUsedCountl	229
9.17.3.16 chMBPeekl	230
9.18 I/O Queues	231
9.18.1 Detailed Description	231
9.18.2 Macro Definition Documentation	233
9.18.2.1 Q_OK	233
9.18.2.2 Q_TIMEOUT	233
9.18.2.3 Q_RESET	233
9.18.2.4 Q_EMPTY	233
9.18.2.5 Q_FULL	233
9.18.2.6 _INPUTQUEUE_DATA	233
9.18.2.7 INPUTQUEUE_DECL	234
9.18.2.8 _OUTPUTQUEUE_DATA	234

9.18.2.9	OUTPUTQUEUE_DECL	234
9.18.2.10	chQSizeX	235
9.18.2.11	chQSpaceI	235
9.18.2.12	chQGetLinkX	235
9.18.3	Typedef Documentation	236
9.18.3.1	io_queue_t	236
9.18.3.2	qnotify_t	236
9.18.3.3	input_queue_t	236
9.18.3.4	output_queue_t	236
9.18.4	Function Documentation	236
9.18.4.1	chIQObjectInit	236
9.18.4.2	chIQResetI	237
9.18.4.3	chIQPutI	238
9.18.4.4	chIQGetTimeout	239
9.18.4.5	chIQReadTimeout	240
9.18.4.6	chOQObjectInit	241
9.18.4.7	chOQResetI	242
9.18.4.8	chOQPutTimeout	243
9.18.4.9	chOQGetI	244
9.18.4.10	chOQWriteTimeout	245
9.18.4.11	chIQGetFullI	246
9.18.4.12	chIQGetEmptyI	247
9.18.4.13	chIQIsEmptyI	247
9.18.4.14	chIQIsFullI	248
9.18.4.15	chIQGet	249
9.18.4.16	chOQGetFullI	249
9.18.4.17	chOQGetEmptyI	250
9.18.4.18	chOQIsEmptyI	251
9.18.4.19	chOQIsFullI	251
9.18.4.20	chOQPut	252
9.19	Memory Management	253
9.19.1	Detailed Description	253
9.20	Core Memory Manager	254
9.20.1	Detailed Description	254
9.20.2	Macro Definition Documentation	255
9.20.2.1	MEM_ALIGN_SIZE	255
9.20.2.2	MEM_ALIGN_MASK	255
9.20.2.3	MEM_ALIGN_PREV	255
9.20.2.4	MEM_ALIGN_NEXT	255
9.20.2.5	MEM_IS_ALIGNED	255

9.20.3	Typedef Documentation	255
9.20.3.1	memgetfunc_t	255
9.20.4	Function Documentation	255
9.20.4.1	_core_init	255
9.20.4.2	chCoreAlloc	255
9.20.4.3	chCoreAllocI	256
9.20.4.4	chCoreGetStatusX	257
9.21	Heaps	258
9.21.1	Detailed Description	258
9.21.2	Typedef Documentation	258
9.21.2.1	memory_heap_t	258
9.21.3	Function Documentation	259
9.21.3.1	_heap_init	259
9.21.3.2	chHeapObjectInit	259
9.21.3.3	chHeapAlloc	260
9.21.3.4	chHeapFree	260
9.21.3.5	chHeapStatus	261
9.21.4	Variable Documentation	261
9.21.4.1	default_heap	261
9.22	Memory Pools	262
9.22.1	Detailed Description	262
9.22.2	Macro Definition Documentation	263
9.22.2.1	_MEMORYPOOL_DATA	263
9.22.2.2	MEMORYPOOL_DECL	263
9.22.3	Function Documentation	263
9.22.3.1	chPoolObjectInit	263
9.22.3.2	chPoolLoadArray	263
9.22.3.3	chPoolAllocI	264
9.22.3.4	chPoolAlloc	265
9.22.3.5	chPoolFreel	265
9.22.3.6	chPoolFree	266
9.22.3.7	chPoolAdd	267
9.22.3.8	chPoolAddI	268
9.23	Dynamic Threads	269
9.23.1	Detailed Description	269
9.23.2	Function Documentation	269
9.23.2.1	chThdAddRef	269
9.23.2.2	chThdRelease	270
9.23.2.3	chThdCreateFromHeap	271
9.23.2.4	chThdCreateFromMemoryPool	272

9.24 Streams and Files	274
9.24.1 Detailed Description	274
9.25 Abstract Sequential Streams	275
9.25.1 Detailed Description	275
9.25.2 Macro Definition Documentation	275
9.25.2.1 <code>_base_sequential_stream_methods</code>	275
9.25.2.2 <code>_base_sequential_stream_data</code>	276
9.25.2.3 <code>chSequentialStreamWrite</code>	276
9.25.2.4 <code>chSequentialStreamRead</code>	276
9.25.2.5 <code>chSequentialStreamPut</code>	276
9.25.2.6 <code>chSequentialStreamGet</code>	277
9.26 Registry	278
9.26.1 Detailed Description	278
9.26.2 Macro Definition Documentation	279
9.26.2.1 <code>REG_REMOVE</code>	279
9.26.2.2 <code>REG_INSERT</code>	279
9.26.3 Function Documentation	279
9.26.3.1 <code>chRegFirstThread</code>	279
9.26.3.2 <code>chRegNextThread</code>	280
9.26.3.3 <code>chRegSetThreadName</code>	281
9.26.3.4 <code>chRegGetThreadNameX</code>	281
9.26.3.5 <code>chRegSetThreadNameX</code>	282
9.27 Debug	283
9.27.1 Detailed Description	283
9.27.2 Macro Definition Documentation	285
9.27.2.1 <code>CH_DBG_TRACE_BUFFER_SIZE</code>	285
9.27.2.2 <code>CH_DBG_STACK_FILL_VALUE</code>	285
9.27.2.3 <code>CH_DBG_THREAD_FILL_VALUE</code>	285
9.27.2.4 <code>chDbgCheck</code>	285
9.27.2.5 <code>chDbgAssert</code>	285
9.27.3 Function Documentation	286
9.27.3.1 <code>_dbg_check_disable</code>	286
9.27.3.2 <code>_dbg_check_suspend</code>	286
9.27.3.3 <code>_dbg_check_enable</code>	287
9.27.3.4 <code>_dbg_check_lock</code>	287
9.27.3.5 <code>_dbg_check_unlock</code>	287
9.27.3.6 <code>_dbg_check_lock_from_isr</code>	288
9.27.3.7 <code>_dbg_check_unlock_from_isr</code>	288
9.27.3.8 <code>_dbg_check_enter_isr</code>	288
9.27.3.9 <code>_dbg_check_leave_isr</code>	289

9.27.3.10 chDbgCheckClassI	289
9.27.3.11 chDbgCheckClassS	290
9.27.3.12 _dbg_trace_init	290
9.27.3.13 _dbg_trace	290
9.28 Time Measurement	292
9.28.1 Detailed Description	292
9.28.2 Function Documentation	292
9.28.2.1 _tm_init	292
9.28.2.2 chTMOBJECTINIT	293
9.28.2.3 chTMStartMeasurementX	294
9.28.2.4 chTMStopMeasurementX	294
9.28.2.5 chTMChainMeasurementToX	294
9.29 Statistics	295
9.29.1 Detailed Description	295
9.29.2 Function Documentation	295
9.29.2.1 _stats_init	295
9.29.2.2 _stats_increase_irq	296
9.29.2.3 _stats_ctxswc	296
9.29.2.4 _stats_start_measure_crit_thd	296
9.29.2.5 _stats_stop_measure_crit_thd	297
9.29.2.6 _stats_start_measure_crit_isr	297
9.29.2.7 _stats_stop_measure_crit_isr	297
9.30 Port Layer	298
9.30.1 Detailed Description	298
9.30.2 Macro Definition Documentation	299
9.30.2.1 PORT_ARCHITECTURE_XXX	299
9.30.2.2 PORT_ARCHITECTURE_XXX_YYY	300
9.30.2.3 PORT_ARCHITECTURE_NAME	300
9.30.2.4 PORT_COMPILER_NAME	300
9.30.2.5 PORT_SUPPORTS_RT	300
9.30.2.6 PORT_INFO	300
9.30.2.7 PORT_IDLE_THREAD_STACK_SIZE	300
9.30.2.8 PORT_INT_REQUIRED_STACK	300
9.30.2.9 PORT_USE_ALT_TIMER	300
9.30.2.10 PORT_SETUP_CONTEXT	300
9.30.2.11 PORT_WA_SIZE	301
9.30.2.12 PORT_IRQ_IS_VALID_PRIORITY	301
9.30.2.13 PORT_IRQ_IS_VALID_KERNEL_PRIORITY	301
9.30.2.14 PORT_IRQ_PROLOGUE	301
9.30.2.15 PORT_IRQ_EPILOGUE	301

9.30.2.16 PORT_IRQ_HANDLER	301
9.30.2.17 PORT_FAST_IRQ_HANDLER	301
9.30.2.18 port_switch	302
9.30.3 TYPEDOC Documentation	302
9.30.3.1 stkalign_t	302
9.30.4 FUNCTION Documentation	302
9.30.4.1 _port_init	302
9.30.4.2 _port_switch	302
9.30.4.3 port_get_irq_status	302
9.30.4.4 port_irq_enabled	303
9.30.4.5 port_is_isr_context	303
9.30.4.6 port_lock	303
9.30.4.7 port_unlock	303
9.30.4.8 port_lock_from_isr	303
9.30.4.9 port_unlock_from_isr	303
9.30.4.10 port_disable	304
9.30.4.11 port_suspend	304
9.30.4.12 port_enable	304
9.30.4.13 port_wait_for_interrupt	304
9.30.4.14 port_rt_get_counter_value	304
9.31 TEST Runtime	305
9.31.1 Detailed Description	305
9.31.2 Macro Definition Documentation	306
9.31.2.1 DELAY_BETWEEN_TESTS	306
9.31.2.2 TEST_NO_BENCHMARKS	306
9.31.2.3 test_fail	306
9.31.2.4 test_assert	306
9.31.2.5 test_assert_lock	306
9.31.2.6 test_assert_sequence	307
9.31.2.7 test_assert_time_window	307
9.31.3 Function Documentation	307
9.31.3.1 test_printf	307
9.31.3.2 test_print	307
9.31.3.3 test_printfln	308
9.31.3.4 test_emit_token	308
9.31.3.5 test_terminate_threads	309
9.31.3.6 test_wait_threads	309
9.31.3.7 test_wait_tick	309
9.31.3.8 test_start_timer	310
9.31.3.9 TestThread	310

9.31.4 Variable Documentation	311
9.31.4.1 test_timer_done	311
9.32 Customer	312
9.32.1 Detailed Description	312
9.32.2 Macro Definition Documentation	312
9.32.2.1 CH_CUSTOMER_ID_STRING	312
9.32.2.2 CH_CUSTOMER_ID_CODE	312
9.33 License	313
9.33.1 Detailed Description	313
9.33.2 Macro Definition Documentation	313
9.33.2.1 CH_LICENSE	313
9.33.2.2 CH_LICENSE_TYPE_STRING	314
9.33.2.3 CH_LICENSE_ID_STRING	314
9.33.2.4 CH_LICENSE_ID_CODE	314
9.33.2.5 CH_LICENSE_MODIFIABLE_CODE	314
9.33.2.6 CH_LICENSE_FEATURES	314
9.33.2.7 CH_LICENSE_MAX_DEPLOY	314
10 Data Structure Documentation	315
10.1 BaseSequentialStream Struct Reference	315
10.1.1 Detailed Description	315
10.1.2 Field Documentation	316
10.1.2.1 vmt	316
10.2 BaseSequentialStreamVMT Struct Reference	316
10.2.1 Detailed Description	316
10.3 binary_semaphore_t Struct Reference	316
10.3.1 Detailed Description	318
10.4 ch_mutex Struct Reference	318
10.4.1 Detailed Description	320
10.4.2 Field Documentation	320
10.4.2.1 m_queue	320
10.4.2.2 m_owner	320
10.4.2.3 m_next	320
10.4.2.4 m_cnt	320
10.5 ch_semaphore Struct Reference	320
10.5.1 Detailed Description	321
10.5.2 Field Documentation	321
10.5.2.1 s_queue	321
10.5.2.2 s_cnt	322
10.6 ch_swcm_event_t Struct Reference	322

10.6.1	Detailed Description	323
10.6.2	Field Documentation	323
10.6.2.1	se_time	323
10.6.2.2	se_tp	323
10.6.2.3	se_wtobjp	323
10.6.2.4	se_state	323
10.7	ch_system Struct Reference	323
10.7.1	Detailed Description	324
10.7.2	Member Function Documentation	324
10.7.2.1	THD_WORKING_AREA	324
10.7.3	Field Documentation	324
10.7.3.1	rlist	324
10.7.3.2	vtlist	324
10.7.3.3	dbg	324
10.7.3.4	mainthread	324
10.7.3.5	tm	325
10.7.3.6	kernel_stats	325
10.8	ch_system_debug Struct Reference	325
10.8.1	Detailed Description	326
10.8.2	Field Documentation	326
10.8.2.1	panic_msg	326
10.8.2.2	isr_cnt	326
10.8.2.3	lock_cnt	326
10.8.2.4	trace_buffer	326
10.9	ch_thread Struct Reference	326
10.9.1	Detailed Description	329
10.9.2	Field Documentation	329
10.9.2.1	p_next	329
10.9.2.2	p_prev	329
10.9.2.3	p_prio	329
10.9.2.4	p_ctx	329
10.9.2.5	p_newer	329
10.9.2.6	p_older	329
10.9.2.7	p_name	329
10.9.2.8	p_stklimit	329
10.9.2.9	p_state	329
10.9.2.10	p_flags	329
10.9.2.11	p_refs	330
10.9.2.12	p_preempt	330
10.9.2.13	p_time	330

10.9.2.14 rdymsg	330
10.9.2.15 exitcode	330
10.9.2.16 wtobjp	330
10.9.2.17 wttrp	330
10.9.2.18 wtsemp	331
10.9.2.19 wtmtxp	331
10.9.2.20 ewmask	331
10.9.2.21 p_u	331
10.9.2.22 p_waiting	331
10.9.2.23 p_msgqueue	331
10.9.2.24 p_msg	331
10.9.2.25 p_epending	331
10.9.2.26 p_mtxlist	332
10.9.2.27 p_realprio	332
10.9.2.28 p_mpool	332
10.9.2.29 p_stats	332
10.10ch_threads_list Struct Reference	332
10.10.1 Detailed Description	333
10.10.2 Field Documentation	334
10.10.2.1 p_next	334
10.11ch_threads_queue Struct Reference	334
10.11.1 Detailed Description	335
10.11.2 Field Documentation	335
10.11.2.1 p_next	335
10.11.2.2 p_prev	335
10.12ch_trace_buffer_t Struct Reference	335
10.12.1 Detailed Description	336
10.12.2 Field Documentation	337
10.12.2.1 tb_size	337
10.12.2.2 tb_ptr	337
10.12.2.3 tb_buffer	337
10.13ch_virtual_timer Struct Reference	337
10.13.1 Detailed Description	338
10.13.2 Field Documentation	338
10.13.2.1 vt_next	338
10.13.2.2 vt_prev	338
10.13.2.3 vt_delta	339
10.13.2.4 vt_func	339
10.13.2.5 vt_par	339
10.14ch_virtual_timers_list Struct Reference	339

10.14.1 Detailed Description	340
10.14.2 Field Documentation	340
10.14.2.1 vt_next	340
10.14.2.2 vt_prev	341
10.14.2.3 vt_delta	341
10.14.2.4 vt_systime	341
10.14.2.5 vt_lasttime	341
10.15chdebug_t Struct Reference	341
10.15.1 Detailed Description	342
10.15.2 Field Documentation	342
10.15.2.1 ch_identifier	342
10.15.2.2 ch_zero	342
10.15.2.3 ch_size	342
10.15.2.4 ch_version	343
10.15.2.5 ch_ptrsize	343
10.15.2.6 ch_timesize	343
10.15.2.7 ch_threadsize	343
10.15.2.8 cf_off_prio	343
10.15.2.9 cf_off_ctx	343
10.15.2.10cf_off_newer	343
10.15.2.11cf_off_older	343
10.15.2.12cf_off_name	343
10.15.2.13cf_off_stklimit	343
10.15.2.14cf_off_state	343
10.15.2.15cf_off_flags	343
10.15.2.16cf_off_refs	344
10.15.2.17cf_off_preempt	344
10.15.2.18cf_off_time	344
10.16condition_variable Struct Reference	344
10.16.1 Detailed Description	345
10.16.2 Field Documentation	345
10.16.2.1 c_queue	345
10.17context Struct Reference	345
10.17.1 Detailed Description	345
10.18event_listener Struct Reference	345
10.18.1 Detailed Description	346
10.18.2 Field Documentation	346
10.18.2.1 el_next	346
10.18.2.2 el_listener	347
10.18.2.3 el_events	347

10.18.2.4 el_flags	347
10.18.2.5 el_wflags	347
10.19 event_source Struct Reference	347
10.19.1 Detailed Description	348
10.19.2 Field Documentation	348
10.19.2.1 es_next	348
10.20 heap_header Union Reference	349
10.20.1 Detailed Description	349
10.20.2 Field Documentation	349
10.20.2.1 next	349
10.20.2.2 heap	349
10.20.2.3 u	349
10.20.2.4 size	349
10.21 io_queue Struct Reference	350
10.21.1 Detailed Description	351
10.21.2 Field Documentation	351
10.21.2.1 q_waiting	351
10.21.2.2 q_counter	351
10.21.2.3 q_buffer	351
10.21.2.4 q_top	351
10.21.2.5 q_wptr	351
10.21.2.6 q_rptr	351
10.21.2.7 q_notify	351
10.21.2.8 q_link	351
10.22 kernel_stats_t Struct Reference	351
10.22.1 Detailed Description	352
10.22.2 Field Documentation	352
10.22.2.1 n_irq	352
10.22.2.2 n_ctxswc	352
10.22.2.3 m_crit_thd	353
10.22.2.4 m_crit_isr	353
10.23 mailbox_t Struct Reference	353
10.23.1 Detailed Description	354
10.23.2 Field Documentation	354
10.23.2.1 mb_buffer	354
10.23.2.2 mb_top	354
10.23.2.3 mb_wptr	354
10.23.2.4 mb_rptr	354
10.23.2.5 mb_fullsem	354
10.23.2.6 mb_emptysem	354

10.24memory_heap Struct Reference	354
10.24.1 Detailed Description	355
10.24.2 Field Documentation	356
10.24.2.1 h_provider	356
10.24.2.2 h_free	356
10.24.2.3 h_mtx	356
10.25memory_pool_t Struct Reference	356
10.25.1 Detailed Description	357
10.25.2 Field Documentation	357
10.25.2.1 mp_next	357
10.25.2.2 mp_object_size	357
10.25.2.3 mp_provider	357
10.26pool_header Struct Reference	357
10.26.1 Detailed Description	357
10.26.2 Field Documentation	357
10.26.2.1 ph_next	357
10.27port_extctx Struct Reference	358
10.27.1 Detailed Description	358
10.28port_intctx Struct Reference	358
10.28.1 Detailed Description	358
10.29 testcase Struct Reference	359
10.29.1 Detailed Description	359
10.29.2 Field Documentation	359
10.29.2.1 name	359
10.29.2.2 setup	359
10.29.2.3 teardown	360
10.29.2.4 execute	360
10.30time_measurement_t Struct Reference	360
10.30.1 Detailed Description	360
10.30.2 Field Documentation	361
10.30.2.1 best	361
10.30.2.2 worst	361
10.30.2.3 last	361
10.30.2.4 n	361
10.30.2.5 cumulative	361
10.31tm_calibration_t Struct Reference	361
10.31.1 Detailed Description	362
10.31.2 Field Documentation	362
10.31.2.1 offset	362

11 File Documentation	363
11.1 ch.h File Reference	363
11.1.1 Detailed Description	364
11.2 chbsem.h File Reference	364
11.2.1 Detailed Description	365
11.3 chcond.c File Reference	365
11.3.1 Detailed Description	365
11.4 chcond.h File Reference	365
11.4.1 Detailed Description	366
11.5 chconf.h File Reference	366
11.5.1 Detailed Description	368
11.6 chcore.c File Reference	368
11.6.1 Detailed Description	369
11.7 chcore.h File Reference	369
11.7.1 Detailed Description	370
11.8 chcustomer.h File Reference	371
11.8.1 Detailed Description	371
11.9 chdebug.c File Reference	371
11.9.1 Detailed Description	372
11.10chdebug.h File Reference	372
11.10.1 Detailed Description	372
11.11chdynamic.c File Reference	372
11.11.1 Detailed Description	373
11.12chdynamic.h File Reference	373
11.12.1 Detailed Description	373
11.13chevents.c File Reference	373
11.13.1 Detailed Description	374
11.14chevents.h File Reference	374
11.14.1 Detailed Description	376
11.15chheap.c File Reference	376
11.15.1 Detailed Description	376
11.16chheap.h File Reference	377
11.16.1 Detailed Description	377
11.17chlicense.h File Reference	377
11.17.1 Detailed Description	378
11.18chmboxes.c File Reference	378
11.18.1 Detailed Description	379
11.19chmboxes.h File Reference	379
11.19.1 Detailed Description	380
11.20chmemcore.c File Reference	380

11.20.1 Detailed Description	380
11.21 chmemcore.h File Reference	380
11.21.1 Detailed Description	381
11.22 chmempools.c File Reference	381
11.22.1 Detailed Description	382
11.23 chmempools.h File Reference	382
11.23.1 Detailed Description	383
11.24 chmsg.c File Reference	383
11.24.1 Detailed Description	383
11.25 chmsg.h File Reference	383
11.25.1 Detailed Description	383
11.26 chmtx.c File Reference	384
11.26.1 Detailed Description	384
11.27 chmtx.h File Reference	384
11.27.1 Detailed Description	385
11.28 chqueues.c File Reference	385
11.28.1 Detailed Description	386
11.29 chqueues.h File Reference	386
11.29.1 Detailed Description	388
11.30 chregistry.c File Reference	388
11.30.1 Detailed Description	388
11.31 chregistry.h File Reference	388
11.31.1 Detailed Description	389
11.32 chschd.c File Reference	389
11.32.1 Detailed Description	390
11.33 chschd.h File Reference	390
11.33.1 Detailed Description	393
11.34 chsem.c File Reference	393
11.34.1 Detailed Description	394
11.35 chsem.h File Reference	394
11.35.1 Detailed Description	395
11.36 chstats.c File Reference	395
11.36.1 Detailed Description	395
11.37 chstats.h File Reference	396
11.37.1 Detailed Description	396
11.38 chstreams.h File Reference	396
11.38.1 Detailed Description	397
11.39 chsys.c File Reference	397
11.39.1 Detailed Description	397
11.40 chsys.h File Reference	397

11.40.1 Detailed Description	399
11.41 chsystypes.h File Reference	399
11.41.1 Detailed Description	400
11.42 chthreads.c File Reference	400
11.42.1 Detailed Description	401
11.43 chthreads.h File Reference	401
11.43.1 Detailed Description	403
11.44 chtm.c File Reference	403
11.44.1 Detailed Description	403
11.45 chtm.h File Reference	403
11.45.1 Detailed Description	404
11.46 chtypes.h File Reference	404
11.46.1 Detailed Description	405
11.47 chvt.c File Reference	405
11.47.1 Detailed Description	405
11.48 chvt.h File Reference	405
11.48.1 Detailed Description	407
11.49 test.c File Reference	407
11.49.1 Detailed Description	408
11.50 test.h File Reference	408
11.50.1 Detailed Description	409
11.51 testbmk.c File Reference	409
11.51.1 Detailed Description	409
11.51.2 Variable Documentation	409
11.51.2.1 patternbmk	409
11.52 testbmk.h File Reference	410
11.52.1 Detailed Description	410
11.52.2 Variable Documentation	410
11.52.2.1 patternbmk	410
11.53 testdyn.c File Reference	410
11.53.1 Detailed Description	410
11.53.2 Variable Documentation	411
11.53.2.1 patterndyn	411
11.54 testdyn.h File Reference	411
11.54.1 Detailed Description	411
11.54.2 Variable Documentation	411
11.54.2.1 patterndyn	411
11.55 testevt.c File Reference	411
11.55.1 Detailed Description	412
11.55.2 Variable Documentation	412

11.55.2.1 patternevt	412
11.56 testevt.h File Reference	412
11.56.1 Detailed Description	412
11.56.2 Variable Documentation	412
11.56.2.1 patternevt	412
11.57 testheap.c File Reference	412
11.57.1 Detailed Description	413
11.57.2 Variable Documentation	413
11.57.2.1 patternheap	413
11.58 testheap.h File Reference	413
11.58.1 Detailed Description	413
11.58.2 Variable Documentation	413
11.58.2.1 patternheap	413
11.59 testmbox.c File Reference	413
11.59.1 Detailed Description	414
11.59.2 Variable Documentation	414
11.59.2.1 patternmbox	414
11.60 testmbox.h File Reference	414
11.60.1 Detailed Description	414
11.60.2 Variable Documentation	414
11.60.2.1 patternmbox	414
11.61 testmsg.c File Reference	414
11.61.1 Detailed Description	415
11.61.2 Variable Documentation	415
11.61.2.1 patternmsg	415
11.62 testmsg.h File Reference	415
11.62.1 Detailed Description	415
11.62.2 Variable Documentation	415
11.62.2.1 patternmsg	415
11.63 testmtx.c File Reference	415
11.63.1 Detailed Description	416
11.63.2 Variable Documentation	416
11.63.2.1 patternmtx	416
11.64 testmtx.h File Reference	416
11.64.1 Detailed Description	416
11.64.2 Variable Documentation	416
11.64.2.1 patternmtx	416
11.65 testpools.c File Reference	416
11.65.1 Detailed Description	417
11.66 testpools.h File Reference	417

11.66.1 Detailed Description	417
11.67testqueues.c File Reference	417
11.67.1 Detailed Description	417
11.67.2 Variable Documentation	417
11.67.2.1 patternqueues	417
11.68testqueues.h File Reference	417
11.68.1 Detailed Description	418
11.68.2 Variable Documentation	418
11.68.2.1 patternqueues	418
11.69testsem.c File Reference	418
11.69.1 Detailed Description	418
11.69.2 Variable Documentation	418
11.69.2.1 patternsem	418
11.70testsem.h File Reference	418
11.70.1 Detailed Description	419
11.70.2 Variable Documentation	419
11.70.2.1 patternsem	419
11.71testsys.c File Reference	419
11.71.1 Detailed Description	419
11.71.2 Variable Documentation	419
11.71.2.1 patternsys	419
11.72testsys.h File Reference	419
11.72.1 Detailed Description	420
11.72.2 Variable Documentation	420
11.72.2.1 patternsys	420
11.73testthd.c File Reference	420
11.73.1 Detailed Description	420
11.73.2 Variable Documentation	420
11.73.2.1 patternthd	420
11.74testthd.h File Reference	420
11.74.1 Detailed Description	421
11.74.2 Variable Documentation	421
11.74.2.1 patternthd	421

Chapter 1

ChibiOS/RT

1.1 Copyright

Copyright (C) 2006..2015 Giovanni Di Sirio. All rights reserved.

Neither the whole nor any part of the information contained in this document may be adapted or reproduced in any form except with the prior written permission of the copyright holder.

The product described in this document is subject to continuous developments and improvements. All particulars of the product and its use contained in this document are given by Giovanni Di Sirio in good faith. However, all warranties implied or expressed, including but not limited to implied warranties of merchantability, or fitness for purpose, are excluded.

This document is intended only to assist the reader in the use of the product. Giovanni Di Sirio shall not be liable for any loss or damage arising from the use of any information in this document, or any error or omission in such information, or any incorrect use of the product.

1.2 Introduction

This document is the Reference Manual for the ChibiOS/RT portable Kernel.

1.3 Related Documents

- ChibiOS/RT General Architecture

Chapter 2

Kernel Concepts

ChibiOS/RT Kernel Concepts

- [Naming Conventions](#)
- [API Name Suffixes](#)
- [Interrupt Classes](#)
- [System States](#)
- [Scheduling](#)
- [Thread States](#)
- [Priority Levels](#)
- [Thread Working Area](#)

2.1 Naming Conventions

ChibiOS/RT APIs are all named following this convention: *ch<group><action><suffix>()*. The possible groups are: *Sys*, *Sch*, *Time*, *VT*, *Thd*, *Sem*, *Mtx*, *Cond*, *Evt*, *Msg*, *Reg*, *SequentialStream*, *IO*, *IQ*, *OQ*, *Dbg*, *Core*, *Heap*, *Pool*.

2.2 API Name Suffixes

The suffix can be one of the following:

- **None**, APIs without any suffix can be invoked only from the user code in the **Normal** state unless differently specified. See [System States](#).
- "**I**", I-Class APIs are invokable only from the **I-Locked** or **S-Locked** states. See [System States](#).
- "**S**", S-Class APIs are invokable only from the **S-Locked** state. See [System States](#).

Examples: `chThdCreateStatic()`, `chSemSignalI()`, `chIQGetTimeout()`.

2.3 Interrupt Classes

In ChibiOS/RT there are three logical interrupt classes:

- **Regular Interrupts.** Maskable interrupt sources that cannot preempt (small parts of) the kernel code and are thus able to invoke operating system APIs from within their handlers. The interrupt handlers belonging to this class must be written following some rules. See the system APIs group and the web article [How to write interrupt handlers](#).
- **Fast Interrupts.** Maskable interrupt sources with the ability to preempt the kernel code and thus have a lower latency and are less subject to jitter, see the web article [Response Time and Jitter](#). Such sources are not supported on all the architectures.
Fast interrupts are not allowed to invoke any operating system API from within their handlers. Fast interrupt sources may, however, pend a lower priority regular interrupt where access to the operating system is possible.
- **Non Maskable Interrupts.** Non maskable interrupt sources are totally out of the operating system control and have the lowest latency. Such sources are not supported on all the architectures.

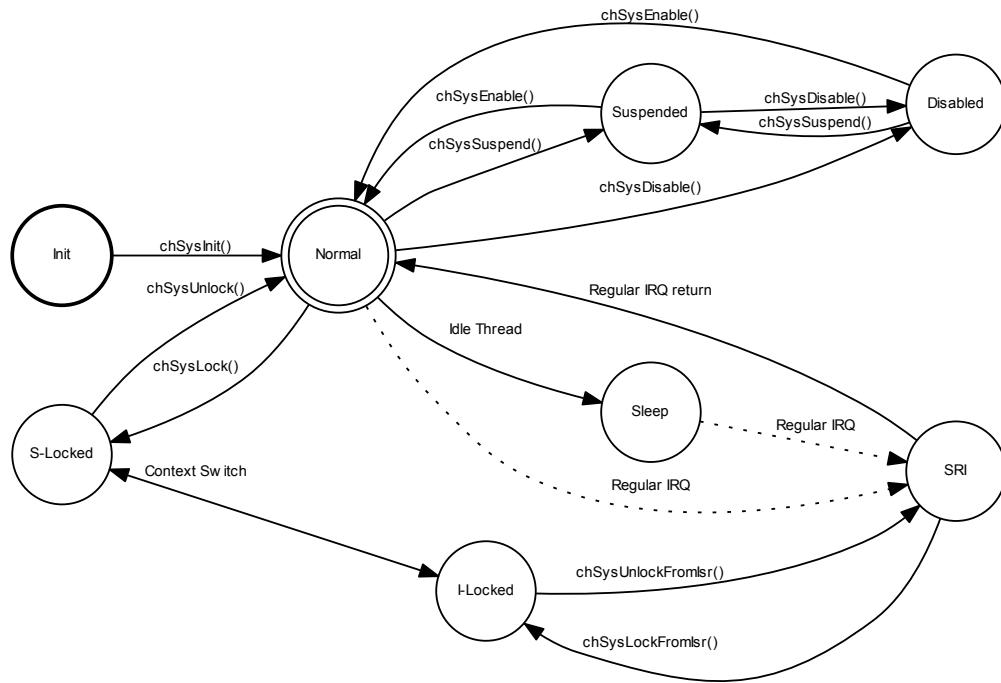
The mapping of the above logical classes into physical interrupts priorities is, of course, port dependent. See the documentation of the various ports for details.

2.4 System States

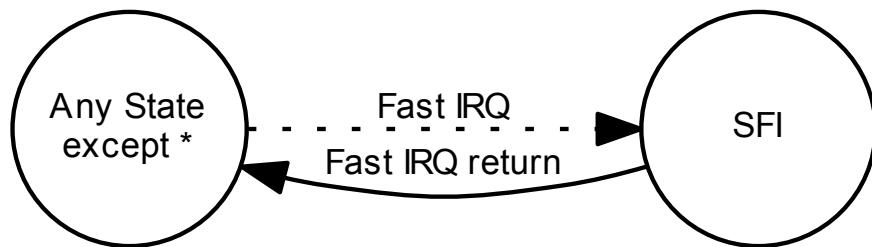
When using ChibiOS/RT the system can be in one of the following logical operating states:

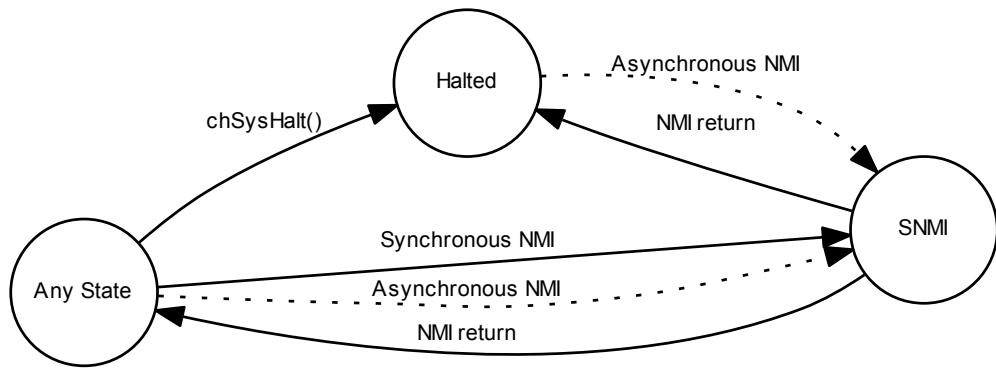
- **Init.** When the system is in this state all the maskable interrupt sources are disabled. In this state it is not possible to use any system API except `chSysInit()`. This state is entered after a physical reset.
- **Normal.** All the interrupt sources are enabled and the system APIs are accessible, threads are running.
- **Suspended.** In this state the fast interrupt sources are enabled but the regular interrupt sources are not. In this state it is not possible to use any system API except `chSysDisable()` or `chSysEnable()` in order to change state.
- **Disabled.** When the system is in this state both the maskable regular and fast interrupt sources are disabled. In this state it is not possible to use any system API except `chSysSuspend()` or `chSysEnable()` in order to change state.
- **Sleep.** Architecture-dependent low power mode, the idle thread goes in this state and waits for interrupts, after servicing the interrupt the Normal state is restored and the scheduler has a chance to reschedule.
- **S-Locked.** Kernel locked and regular interrupt sources disabled. Fast interrupt sources are enabled. [S-Class](#) and [I-Class](#) APIs are invokable in this state.
- **I-Locked.** Kernel locked and regular interrupt sources disabled. [I-Class](#) APIs are invokable from this state.
- **Serving Regular Interrupt.** No system APIs are accessible but it is possible to switch to the I-Locked state using `chSysLockFromIsr()` and then invoke any [I-Class](#) API. Interrupt handlers can be preemptable on some architectures thus is important to switch to I-Locked state before invoking system APIs.
- **Serving Fast Interrupt.** System APIs are not accessible.
- **Serving Non-Maskable Interrupt.** System APIs are not accessible.
- **Halted.** All interrupt sources are disabled and system stopped into an infinite loop. This state can be reached if the debug mode is activated **and** an error is detected **or** after explicitly invoking `chSysHalt()`.

Note that the above states are just **Logical States** that may have no real associated machine state on some architectures. The following diagram shows the possible transitions between the states:



Note, the **SFI**, **Halted** and **SNMI** states were not shown because those are reachable from most states:



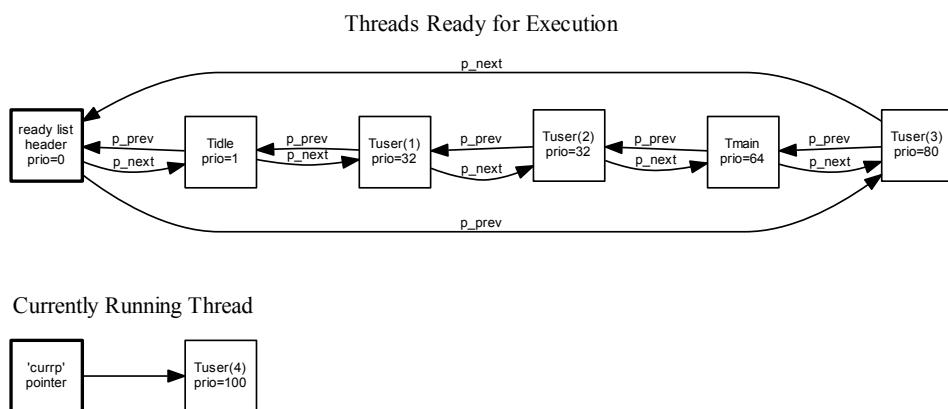


Attention

* except: **Init, Halt, SNMI, Disabled**.

2.5 Scheduling

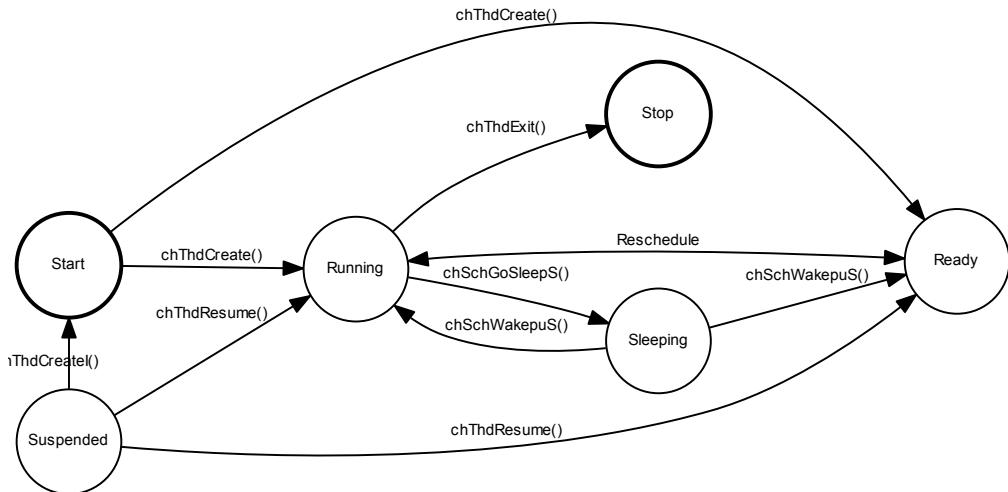
The strategy is very simple the currently ready thread with the highest priority is executed. If more than one thread with equal priority are eligible for execution then they are executed in a round-robin way, the CPU time slice constant is configurable. The ready list is a double linked list of threads ordered by priority.



Note that the currently running thread is not in the ready list, the list only contains the threads ready to be executed but still actually waiting.

2.6 Thread States

The image shows how threads can change their state in ChibiOS/RT.



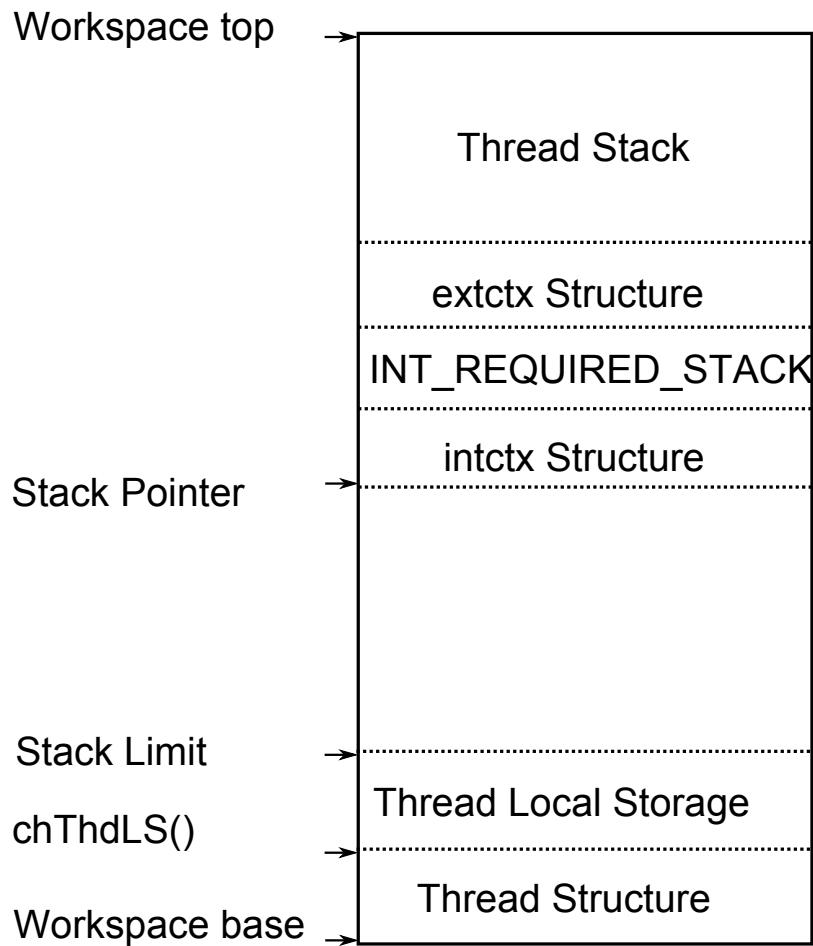
2.7 Priority Levels

Priorities in ChibiOS/RT are a contiguous numerical range but the initial and final values are not enforced. The following table describes the various priority boundaries (from lowest to highest):

- `IDLEPRIOR`, this is the lowest priority level and is reserved for the idle thread, no other threads should share this priority level. This is the lowest numerical value of the priorities space.
- `LOWPRIOR`, the lowest priority level that can be assigned to an user thread.
- `NORMALPRIOR`, this is the central priority level for user threads. It is advisable to assign priorities to threads as values relative to `NORMALPRIOR`, as example `NORMALPRIOR-1` or `NORMALPRIOR+4`, this ensures the portability of code should the numerical range change in future implementations.
- `HIGHPRIOR`, the highest priority level that can be assigned to an user thread.
- `ABSPRIOR`, absolute maximum software priority level, it can be higher than `HIGHPRIOR` but the numerical values above `HIGHPRIOR` up to `ABSPRIOR` (inclusive) are reserved. This is the highest numerical value of the priorities space.

2.8 Thread Working Area

Each thread has its own stack, a Thread structure and some preemption areas. All the structures are allocated into a "Thread Working Area", a thread private heap, usually statically declared in your code. Threads do not use any memory outside the allocated working area except when accessing static shared data.



Note that the preemption area is only present when the thread is not running (switched out), the context switching is done by pushing the registers on the stack of the switched-out thread and popping the registers of the switched-in thread. The preemption area can be divided in up to three structures:

- External Context.
- Interrupt Stack.
- Internal Context.

See the port documentation for details, the area may change on the various ports and some structures may not be present (or be zero-sized).

Chapter 3

Testing Strategy

Description

Most of the ChibiOS/RT demos link a set of software modules (test suite) in order to verify the proper working of the kernel, the port and the demo itself.

Kernel Test Suite

The kernel test suite is divided in modules or test sequences. Each Test Module performs a series of tests on a specified kernel subsystem or subsystems and can report a failure/success status and/or a performance index as the test suite output.

The test suite is usually activated in the demo applications by pressing a button on the target board, see the readme file into the various demos directories. The test suite output is usually sent through a serial port and can be examined by using a terminal emulator program.

Kernel Test Modules

- [Threads and Scheduler test](#)
- [Dynamic APIs test](#)
- [Messages test](#)
- [Semaphores test](#)
- [Mutexes test](#)
- [Events test](#)
- [Mailboxes test](#)
- [I/O Queues test](#)
- [Memory Heap test](#)
- [Memory Pools test](#)
- [System test](#)
- [Kernel Benchmarks](#)

3.1 Threads and Scheduler test

File: [testthd.c](#)

Description

This module implements the test sequence for the [Scheduler](#), [Threads](#) and [Time and Virtual Timers](#) subsystems. Note that the tests on those subsystems are formally required but most of their functionality is already demonstrated because the test suite itself depends on them, anyway double check is good.

Objective

Objective of the test module is to cover 100% of the subsystems code.

Preconditions

None.

Test Cases

- [Ready List functionality #1](#)
- [Ready List functionality #2](#)
- [Threads priority change test](#)
- [Threads delays test](#)

3.1.1 Ready List functionality #1

Description

Five threads, with increasing priority, are enqueued in the ready list and atomically executed. The test expects the threads to perform their operations in increasing priority order regardless of the initial order.

3.1.2 Ready List functionality #2

Description

Five threads, with pseudo-random priority, are enqueued in the ready list and atomically executed. The test expects the threads to perform their operations in increasing priority order regardless of the initial order.

3.1.3 Threads priority change test

Description

A series of priority changes are performed on the current thread in order to verify that the priority change happens as expected.

If the `CH_CFG_USE_MUTEXES` option is enabled then the priority changes are also tested under priority inheritance boosted priority state.

3.1.4 Threads delays test

Description

Delay APIs and associated macros are tested, the invoking thread is verified to wake up at the exact expected time.

3.2 Dynamic APIs test

File: [testdyn.c](#)

Description

This module implements the test sequence for the dynamic thread creation APIs.

Objective

Objective of the test module is to cover 100% of the dynamic APIs code.

Preconditions

The module requires the following kernel options:

- CH_CFG_USE_DYNAMIC
- CH_CFG_USE_HEAP
- CH_CFG_USE_MEMPOOLS

In case some of the required options are not enabled then some or all tests may be skipped.

Test Cases

- [Threads creation from Memory Heap](#)
- [Threads creation from Memory Pool](#)
- [Registry and References test](#)

3.2.1 Threads creation from Memory Heap

Description

Two threads are started by allocating the memory from the Memory Heap then the remaining heap space is arbitrarily allocated and a third tread startup is attempted.

The test expects the first two threads to successfully start and the last one to fail.

3.2.2 Threads creation from Memory Pool

Description

Five thread creation are attempted from a pool containing only four elements.

The test expects the first four threads to successfully start and the last one to fail.

3.2.3 Registry and References test

Description

Registry and Thread References APIs are tested for functionality and coverage.

3.3 Messages test

File: [testmsg.c](#)

Description

This module implements the test sequence for the [Synchronous Messages](#) subsystem.

Objective

Objective of the test module is to cover 100% of the [Synchronous Messages](#) subsystem code.

Preconditions

The module requires the following kernel options:

- CH_CFG_USE_MESSAGES

In case some of the required options are not enabled then some or all tests may be skipped.

Test Cases

- [Messages Server loop](#)

3.3.1 Messages Server loop

Description

A thread is spawned that sends four messages back to the tester thread.

The test expect to receive the messages in the correct sequence and to not find a fifth message waiting.

3.4 Semaphores test

File: [testsem.c](#)

Description

This module implements the test sequence for the [Counting Semaphores](#) subsystem.

Objective

Objective of the test module is to cover 100% of the [Counting Semaphores](#) code.

Preconditions

The module requires the following kernel options:

- CH_CFG_USE_SEMAPHORES

In case some of the required options are not enabled then some or all tests may be skipped.

Test Cases

- Enqueuing test
- Timeout test
- Atomic signal-wait test
- Binary Wait and Signal

3.4.1 Enqueuing test

Description

Five threads with randomized priorities are enqueued to a semaphore then awakened one at time. The test expects that the threads reach their goal in FIFO order or priority order depending on the CH_CFG_US_E_SEMAPHORES_PRIORITY configuration setting.

3.4.2 Timeout test

Description

The three possible semaphore waiting modes (do not wait, wait with timeout, wait without timeout) are explored. The test expects that the semaphore wait function returns the correct value in each of the above scenario and that the semaphore structure status is correct after each operation.

3.4.3 Atomic signal-wait test

Description

This test case explicitly addresses the `chSemWaitSignal()` function. A thread is created that performs a wait and a signal operations. The tester thread is awakened from an atomic wait/signal operation. The test expects that the semaphore wait function returns the correct value in each of the above scenario and that the semaphore structure status is correct after each operation.

3.4.4 Binary Wait and Signal

Description

This test case tests the binary semaphores functionality. The test both checks the binary semaphore status and the expected status of the underlying counting semaphore.

3.5 Mutexes test

File: [testmtx.c](#)

Description

This module implements the test sequence for the [Mutexes](#) and [Condition Variables](#) subsystems. Tests on those subsystems are particularly critical because the system-wide implications of the Priority Inheritance mechanism.

Objective

Objective of the test module is to cover 100% of the subsystems code.

Preconditions

The module requires the following kernel options:

- CH_CFG_USE_MUTEXES
- CH_CFG_USE_CONDVARS
- CH_DBG_THREADS_PROFILING

In case some of the required options are not enabled then some or all tests may be skipped.

Test Cases

- Priority enqueueing test
- Priority inheritance, simple case
- Priority inheritance, complex case
- Priority return verification
- Mutex status
- Condition Variable signal test
- Condition Variable broadcast test
- Condition Variable priority boost test

3.5.1 Priority enqueueing test

Description

Five threads, with increasing priority, are enqueued on a locked mutex then the mutex is unlocked. The test expects the threads to perform their operations in increasing priority order regardless of the initial order.

3.5.2 Priority inheritance, simple case

Description

Three threads are involved in the classic priority inversion scenario, a medium priority thread tries to starve an high priority thread by blocking a low priority thread into a mutex lock zone.

The test expects the threads to reach their goal in increasing priority order by rearranging their priorities in order to avoid the priority inversion trap.

Scenario

This weird looking diagram should explain what happens in the test case:

```

Time ----> 0      10     20     30     40     50     60     70     80     90     100
    0 .....AL+++++++. .... .2+++++++.AUO-----++++++G...
    1 ..... .+++++++.----+-----++++++++.G.... .
    2 ..... .AL.....++++++AUG.... .

```

Legend:

- 0..2 - Priority levels
- +++ - Running
- - Ready
- ... - Waiting or Terminated
- xL - Lock operation on mutex '`x`'
- xUn - Unlock operation on mutex '`x`' with priority returning to level '`n`'
- G - Goal
- ^ - Priority transition (boost or `return`).

3.5.3 Priority inheritance, complex case

Description

Five threads are involved in the complex priority inversion scenario, please refer to the diagram below for the complete scenario.

The test expects the threads to perform their operations in increasing priority order by rearranging their priorities in order to avoid the priority inversion trap.

Scenario

This weird looking diagram should explain what happens in the test case:

```

Time ----> 0      10     20     30     40     50     60     70     80     90     100    110
0 .....BL+++++-----2+++++----4+++++BU0-----G..
1 .....AL+++2+++++BL-----4-----++++++BU4+++AU1-----G..
2 .....AL-----+-----+-----+-----+-----+-----AUG..
3 .....+-----+-----+-----+-----+-----+-----+-----+G..
4 .....+-----+-----+-----+-----+-----+-----+-----+UGC

```

Legend:
0..4 - Priority levels
+++ - Running
--- - Ready
... - Waiting or Terminated
xL - Lock operation on mutex '*x*'
xUn - Unlock operation on mutex '*x*' with priority returning to level '*n*'
^ - Priority transition (boost or *return*).

3.5.4 Priority return verification

Description

Two threads are spawned that try to lock the mutexes locked by the tester thread with precise timing. The test expects that the priority changes caused by the priority inheritance algorithm happen at the right moment and with the right values.

3.5.5 Mutex status

Description

Various tests on the mutex structure status after performing some lock and unlock operations. The test expects that the internal mutex status is consistent after each operation.

3.5.6 Condition Variable signal test

Description

Five threads take a mutex and then enter a conditional variable queue, the tester thread then proceeds to signal the conditional variable five times atomically.

The test expects the threads to reach their goal in increasing priority order regardless of the initial order.

3.5.7 Condition Variable broadcast test

Description

Five threads take a mutex and then enter a conditional variable queue, the tester thread then proceeds to broadcast the conditional variable.

The test expects the threads to reach their goal in increasing priority order regardless of the initial order.

3.5.8 Condition Variable priority boost test

Description

This test case verifies the priority boost of a thread waiting on a conditional variable queue. It tests this very specific situation in order to complete the code coverage.

3.6 Events test

File: [testevt.c](#)

Description

This module implements the test sequence for the [Event Flags](#) subsystem.

Objective

Objective of the test module is to cover 100% of the [Event Flags](#) subsystem.

Preconditions

The module requires the following kernel options:

- CH_CFG_USE_EVENTS
- CH_CFG_USE_EVENTS_TIMEOUT

In case some of the required options are not enabled then some or all tests may be skipped.

Test Cases

- [Events registration and dispatch](#)
- [Events wait and broadcast](#)
- [Events timeout](#)

3.6.1 Events registration and dispatch

Description

Two event listeners are registered on an event source and then unregistered in the same order. The test expects that the even source has listeners after the registrations and after the first unregistration, then, after the second unregistration, the test expects no more listeners. In the second part the test dispatches three event flags and verifies that the associated event handlers are invoked in LSb-first order.

3.6.2 Events wait and broadcast

Description

In this test the following APIs are independently tested by starting threads that signal/broadcast events after fixed delays:

- `chEvtWaitOne()`
- `chEvtWaitAny()`
- `chEvtWaitAll()`

After each test phase the test verifies that the events have been served at the expected time and that there are no stuck event flags.

3.6.3 Events timeout

Description

In this test the following APIs are let to timeout twice: immediately and after 10ms: In this test the following APIs are independently tested by starting threads that broadcast events after fixed delays:

- `chEvtWaitOneTimeout()`
- `chEvtWaitAnyTimeout()`
- `chEvtWaitAllTimeout()`

After each test phase the test verifies that there are no stuck event flags.

3.7 Mailboxes test

File: [testmbox.c](#)

Description

This module implements the test sequence for the [Mailboxes](#) subsystem.

Objective

Objective of the test module is to cover 100% of the [Mailboxes](#) subsystem code.

Note that the [Mailboxes](#) subsystem depends on the [Counting Semaphores](#) subsystem that has to met its testing objectives as well.

Preconditions

The module requires the following kernel options:

- CH_CFG_USE_MAILBOXES

In case some of the required options are not enabled then some or all tests may be skipped.

Test Cases

- Queuing and timeouts

3.7.1 Queuing and timeouts

Description

Messages are posted/fetched from a mailbox in carefully designed sequences in order to stimulate all the possible code paths inside the mailbox.

The test expects to find a consistent mailbox status after each operation.

3.8 I/O Queues test

File: [testqueues.c](#)

Description

This module implements the test sequence for the [I/O Queues](#) subsystem. The tests are performed by inserting and removing data from queues and by checking both the queues status and the correct sequence of the extracted data.

Objective

Objective of the test module is to cover 100% of the [I/O Queues](#) code.

Note that the [I/O Queues](#) subsystem depends on the [Counting Semaphores](#) subsystem that has to met its testing objectives as well.

Preconditions

The module requires the following kernel options:

- CH_CFG_USE_QUEUES (and dependent options)

In case some of the required options are not enabled then some or all tests may be skipped.

Test Cases

- Input Queues functionality and APIs
- Output Queues functionality and APIs

3.8.1 Input Queues functionality and APIs

Description

This test case tests synchronous and asynchronous operations on an `InputQueue` object including timeouts. The queue state must remain consistent through the whole test.

3.8.2 Output Queues functionality and APIs

Description

This test case tests synchronous and asynchronous operations on an `OutputQueue` object including timeouts. The queue state must remain consistent through the whole test.

3.9 Memory Heap test

File: [testheap.c](#)

Description

This module implements the test sequence for the [Heaps](#) subsystem.

Objective

Objective of the test module is to cover 100% of the [Heaps](#) subsystem.

Preconditions

The module requires the following kernel options:

- `CH_CFG_USE_HEAP`

In case some of the required options are not enabled then some or all tests may be skipped.

Test Cases

- [Allocation and fragmentation test](#)

3.9.1 Allocation and fragmentation test

Description

Series of allocations/deallocations are performed in carefully designed sequences in order to stimulate all the possible code paths inside the allocator.

The test expects to find the heap back to the initial status after each sequence.

3.10 Memory Pools test

File: [testpools.c](#)

Description

This module implements the test sequence for the [Memory Pools](#) subsystem.

Objective

Objective of the test module is to cover 100% of the [Memory Pools](#) code.

Preconditions

The module requires the following kernel options:

- CH_CFG_USE_MEMPOOLS

In case some of the required options are not enabled then some or all tests may be skipped.

Test Cases

- [Allocation and enqueueing test](#)

3.10.1 Allocation and enqueueing test

Description

Five memory blocks are added to a memory pool then removed.

The test expects to find the pool queue in the proper status after each operation.

3.11 System test

File: [testsys.c](#)

Description

This module implements the test sequence for the [System Management](#) subsystem.

Objective

Objective of the test module is to cover 100% of the [System Management](#) subsystem code.

Preconditions

None.

Test Cases

- [Critical zones check](#)
- [Interrupts handling](#)
- [System integrity check](#)

3.11.1 Critical zones check

Description

The critical zones API is invoked for coverage.

3.11.2 Interrupts handling

Description

The interrupts handling API is invoked for coverage.

3.11.3 System integrity check

Description

The [chSysIntegrityCheck\(\)](#) API is invoked in order to asses the state of the system data structures.

3.12 Kernel Benchmarks

File: [testbmk.c](#)

Description

This module implements a series of system benchmarks. The benchmarks are useful as a stress test and as a reference when comparing ChibiOS/RT with similar systems.

Objective

Objective of the test module is to provide a performance index for the most critical system subsystems. The performance numbers allow to discover performance regressions between successive ChibiOS/RT releases.

Preconditions

None.

Test Cases

- [Messages performance #1](#)
- [Messages performance #2](#)
- [Messages performance #3](#)
- [Context Switch performance](#)
- [Threads performance, full cycle](#)
- [Threads performance, create/exit only](#)
- [Mass reschedule performance](#)
- [I/O Round-Robin voluntary reschedule.](#)

- I/O Queues throughput
- Virtual Timers set/reset performance
- Semaphores wait/signal performance
- Mutexes lock/unlock performance
- RAM Footprint

3.12.1 Messages performance #1

Description

A message server thread is created with a lower priority than the client thread, the messages throughput per second is measured and the result printed in the output log.

3.12.2 Messages performance #2

Description

A message server thread is created with an higher priority than the client thread, the messages throughput per second is measured and the result printed in the output log.

3.12.3 Messages performance #3

Description

A message server thread is created with an higher priority than the client thread, four lower priority threads crowd the ready list, the messages throughput per second is measured while the ready list and the result printed in the output log.

3.12.4 Context Switch performance

Description

A thread is created that just performs a `chSchGoSleepS()` into a loop, the thread is awakened as fast as possible by the tester thread.

The Context Switch performance is calculated by measuring the number of iterations after a second of continuous operations.

3.12.5 Threads performance, full cycle

Description

Threads are continuously created and terminated into a loop. A full `chThdCreateStatic()` / `chThdExit()` / `chThdWait()` cycle is performed in each iteration.

The performance is calculated by measuring the number of iterations after a second of continuous operations.

3.12.6 Threads performance, create/exit only

Description

Threads are continuously created and terminated into a loop. A partial `chThdCreateStatic()` / `chThdExit()` cycle is performed in each iteration, the `chThdWait()` is not necessary because the thread is created at an higher priority so there is no need to wait for it to terminate.

The performance is calculated by measuring the number of iterations after a second of continuous operations.

3.12.7 Mass reschedule performance

Description

Five threads are created and atomically rescheduled by resetting the semaphore where they are waiting on. The operation is performed into a continuous loop.

The performance is calculated by measuring the number of iterations after a second of continuous operations.

3.12.8 I/O Round-Robin voluntary reschedule.

Description

Five threads are created at equal priority, each thread just increases a variable and yields.

The performance is calculated by measuring the number of iterations after a second of continuous operations.

3.12.9 I/O Queues throughput

Description

Four bytes are written and then read from an `InputQueue` into a continuous loop.

The performance is calculated by measuring the number of iterations after a second of continuous operations.

3.12.10 Virtual Timers set/reset performance

Description

A virtual timer is set and immediately reset into a continuous loop.

The performance is calculated by measuring the number of iterations after a second of continuous operations.

3.12.11 Semaphores wait/signal performance

Description

A counting semaphore is taken/released into a continuous loop, no Context Switch happens because the counter is always non negative.

The performance is calculated by measuring the number of iterations after a second of continuous operations.

3.12.12 Mutexes lock/unlock performance

Description

A mutex is locked/unlocked into a continuous loop, no Context Switch happens because there are no other threads asking for the mutex.

The performance is calculated by measuring the number of iterations after a second of continuous operations.

3.12.13 RAM Footprint

Description

The memory size of the various kernel objects is printed.

Chapter 4

Deprecated List

Function Class:

globalScope> Global **chMtxQueueNotEmptyS** (mutex_t *mp)

This is an **S-Class** API, this function can be invoked from within a system lock zone by threads only.

Chapter 5

Module Index

5.1 Modules

Here is a list of all modules:

RT Kernel	37
Version Numbers and Identification	38
Configuration	39
Kernel Types	50
Base Kernel Services	53
System Management	54
Scheduler	73
Threads	95
Time and Virtual Timers	123
Synchronization	140
Counting Semaphores	141
Binary Semaphores	156
Mutexes	167
Condition Variables	178
Event Flags	188
Synchronous Messages	209
Mailboxes	215
I/O Queues	231
Memory Management	253
Core Memory Manager	254
Heaps	258
Memory Pools	262
Dynamic Threads	269
Streams and Files	274
Abstract Sequential Streams	275
Registry	278
Debug	283
Time Measurement	292
Statistics	295
Port Layer	298
Test Runtime	305
Customer	312
License	313

Chapter 6

Hierarchical Index

6.1 Class Hierarchy

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

BaseSequentialStream	315
BaseSequentialStreamVMT	316
ch_mutex	318
ch_semaphore	320
binary_semaphore_t	316
ch_swc_event_t	322
ch_system	323
ch_system_debug	325
ch_thread	326
ch_threads_list	332
ch_threads_queue	334
ch_trace_buffer_t	335
ch_virtual_timers_list	339
ch_virtual_timer	337
chdebug_t	341
condition_variable	344
context	345
event_listener	345
event_source	347
heap_header	349
io_queue	350
kernel_stats_t	351
mailbox_t	353
memory_heap	354
memory_pool_t	356
pool_header	357
port_extctx	358
port_intctx	358
testcase	359
time_measurement_t	360
tm_calibration_t	361

Chapter 7

Data Structure Index

7.1 Data Structures

Here are the data structures with brief descriptions:

BaseSequentialStream	Base stream class	315
BaseSequentialStreamVMT	BaseSequentialStream virtual methods table	316
binary_semaphore_t	Binary semaphore type	316
ch_mutex	Mutex structure	318
ch_semaphore	Semaphore structure	320
ch_swc_event_t	Trace buffer record	322
ch_system	System data structure	323
ch_system_debug	System debug data structure	325
ch_thread	Structure representing a thread	326
ch_threads_list	Generic threads single link list, it works like a stack	332
ch_threads_queue	Generic threads bidirectional linked list header and element	334
ch_trace_buffer_t	Trace buffer header	335
ch_virtual_timer	Virtual Timer descriptor structure	337
ch_virtual_timers_list	Virtual timers list header	339
chdebug_t	ChibiOS/RT memory signature record	341
condition_variable	Condition_variable_t structure	344
context	Platform dependent part of the thread_t structure	345
event_listener	Event Listener structure	345
event_source	Event Source structure	347

heap_header		
Memory heap block header		349
io_queue		
Generic I/O queue structure		350
kernel_stats_t		
Type of a kernel statistics structure		351
mailbox_t		
Structure representing a mailbox object		353
memory_heap		
Structure describing a memory heap		354
memory_pool_t		
Memory pool descriptor		356
pool_header		
Memory pool free object header		357
port_extctx		
Interrupt saved context		358
port_intctx		
System saved context		358
testcase		
Structure representing a test case		359
time_measurement_t		
Type of a Time Measurement object		360
tm_calibration_t		
Type of a time measurement calibration data		361

Chapter 8

File Index

8.1 File List

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

ch.h	ChibiOS/RT main include file	363
chbsem.h	Binary semaphores structures and macros	364
chcond.c	Condition Variables code	365
chcond.h	Condition Variables macros and structures	365
chconf.h	Configuration file template	366
chcore.c	Port related template code	368
chcore.h	Port related template macros and structures	369
chcustomer.h	Customer-related info	371
chdebug.c	ChibiOS/RT Debug code	371
chdebug.h	Debug macros and structures	372
chdynamic.c	Dynamic threads code	372
chdynamic.h	Dynamic threads macros and structures	373
chevents.c	Events code	373
chevents.h	Events macros and structures	374
chheap.c	Heaps code	376
chheap.h	Heaps macros and structures	377
chlicense.h	License Module macros and structures	377
chmboxes.c	Mailboxes code	378
chmboxes.h	Mailboxes macros and structures	379

chmemcore.c	
Core memory manager code	380
chmemcore.h	
Core memory manager macros and structures	380
chmpools.c	
Memory Pools code	381
chmpools.h	
Memory Pools macros and structures	382
chmsg.c	
Messages code	383
chmsg.h	
Messages macros and structures	383
chmtx.c	
Mutexes code	384
chmtx.h	
Mutexes macros and structures	384
chqueues.c	
I/O Queues code	385
chqueues.h	
I/O Queues macros and structures	386
chregistry.c	
Threads registry code	388
chregistry.h	
Threads registry macros and structures	388
chsched.c	
Scheduler code	389
chsched.h	
Scheduler macros and structures	390
chsem.c	
Semaphores code	393
chsem.h	
Semaphores macros and structures	394
chstats.c	
Statistics module code	395
chstats.h	
Statistics module macros and structures	396
chstreams.h	
Data streams	396
chsys.c	
System related code	397
chsys.h	
System related macros and structures	397
chsysypes.h	
System types header	399
chthreads.c	
Threads code	400
chthreads.h	
Threads module macros and structures	401
ctm.c	
Time Measurement module code	403
ctm.h	
Time Measurement module macros and structures	403
ctypes.h	
System types template	404
chvt.c	
Time and Virtual Timers module code	405
chvt.h	
Time and Virtual Timers module macros and structures	405

test.c	Tests support code	407
test.h	Tests support header	408
testbmk.c	Kernel Benchmarks source file	409
testbmk.h	Kernel Benchmarks header file	410
testdyn.c	Dynamic thread APIs test source file	410
testdyn.h	Dynamic thread APIs test header file	411
testevt.c	Events test source file	411
testevt.h	Events test header file	412
testheap.c	Heap test source file	412
testheap.h	Heap header file	413
testmbox.c	Mailboxes test source file	413
testmbox.h	Mailboxes header file	414
testmsg.c	Messages test source file	414
testmsg.h	Messages header file	415
testmtx.c	Mutexes and CondVars test source file	415
testmtx.h	Mutexes and CondVars test header file	416
testpools.c	Memory Pools test source file	416
testpools.h	Memory Pools test header file	417
testqueues.c	I/O Queues test source file	417
testqueues.h	I/O Queues test header file	417
testsem.c	Semaphores test source file	418
testsem.h	Semaphores test header file	418
testsys.c	System test source file	419
testsys.h	System header file	419
testthd.c	Threads and Scheduler test source file	420
testthd.h	Threads and Scheduler test header file	420

Chapter 9

Module Documentation

9.1 RT Kernel

9.1.1 Detailed Description

The kernel is the portable part of ChibiOS/RT, this section documents the various kernel subsystems.

Modules

- [Version Numbers and Identification](#)
- [Configuration](#)
- [Kernel Types](#)
- [Base Kernel Services](#)
- [Synchronization](#)
- [Memory Management](#)
- [Streams and Files](#)
- [Registry](#)
- [Debug](#)
- [Time Measurement](#)
- [Statistics](#)
- [Port Layer](#)

9.2 Version Numbers and Identification

9.2.1 Detailed Description

Kernel related info.

Macros

- `#define _CHIBIOS_RT_`
ChibiOS/RT identification macro.
- `#define CH_KERNEL_STABLE 1`
Stable release flag.

ChibiOS/RT version identification

- `#define CH_KERNEL_VERSION "3.1.5"`
Kernel version string.
- `#define CH_KERNEL_MAJOR 3`
Kernel version major number.
- `#define CH_KERNEL_MINOR 1`
Kernel version minor number.
- `#define CH_KERNEL_PATCH 5`
Kernel version patch number.

9.2.2 Macro Definition Documentation

9.2.2.1 `#define _CHIBIOS_RT_`

ChibiOS/RT identification macro.

9.2.2.2 `#define CH_KERNEL_STABLE 1`

Stable release flag.

9.2.2.3 `#define CH_KERNEL_VERSION "3.1.5"`

Kernel version string.

9.2.2.4 `#define CH_KERNEL_MAJOR 3`

Kernel version major number.

9.2.2.5 `#define CH_KERNEL_MINOR 1`

Kernel version minor number.

9.2.2.6 `#define CH_KERNEL_PATCH 5`

Kernel version patch number.

9.3 Configuration

9.3.1 Detailed Description

Kernel related settings and hooks.

System timers settings

- `#define CH_CFG_ST_RESOLUTION 32`
System time counter resolution.
- `#define CH_CFG_ST_FREQUENCY 10000`
System tick frequency.
- `#define CH_CFG_ST_TIMEDELTA 2`
Time delta constant for the tick-less mode.

Kernel parameters and options

- `#define CH_CFG_TIME_QUANTUM 0`
Round robin interval.
- `#define CH_CFG_MEMCORE_SIZE 0`
Managed RAM size.
- `#define CH_CFG_NO_IDLE_THREAD FALSE`
Idle thread automatic spawn suppression.

Performance options

- `#define CH_CFG_OPTIMIZE_SPEED TRUE`
OS optimization.

Subsystem options

- `#define CH_CFG_USE_TM TRUE`
Time Measurement APIs.
- `#define CH_CFG_USE_REGISTRY TRUE`
Threads registry APIs.
- `#define CH_CFG_USE_WAITEXIT TRUE`
Threads synchronization APIs.
- `#define CH_CFG_USE_SEMAPHORES TRUE`
Semaphores APIs.
- `#define CH_CFG_USE_SEMAPHORES_PRIORITY FALSE`
Semaphores queuing mode.
- `#define CH_CFG_USE_MUTEXES TRUE`
Mutexes APIs.
- `#define CH_CFG_USE_MUTEXES_RECURSIVE FALSE`
Enables recursive behavior on mutexes.
- `#define CH_CFG_USE_CONDVARNS TRUE`
Conditional Variables APIs.
- `#define CH_CFG_USE_CONDVARNS_TIMEOUT TRUE`
Conditional Variables APIs with timeout.
- `#define CH_CFG_USE_EVENTS TRUE`

- `#define CH_CFG_USE_EVENTS_TIMEOUT TRUE`
 - Events Flags APIs.*
 - Events Flags APIs with timeout.*
- `#define CH_CFG_USE_MESSAGES TRUE`
 - Synchronous Messages APIs.*
- `#define CH_CFG_USE_MESSAGES_PRIORITY FALSE`
 - Synchronous Messages queuing mode.*
- `#define CH_CFG_USE_MAILBOXES TRUE`
 - Mailboxes APIs.*
- `#define CH_CFG_USE_QUEUES TRUE`
 - I/O Queues APIs.*
- `#define CH_CFG_USE_MEMCORE TRUE`
 - Core Memory Manager APIs.*
- `#define CH_CFG_USE_HEAP TRUE`
 - Heap Allocator APIs.*
- `#define CH_CFG_USE_MEMPOOLS TRUE`
 - Memory Pools Allocator APIs.*
- `#define CH_CFG_USE_DYNAMIC TRUE`
 - Dynamic Threads APIs.*

Debug options

- `#define CH_DBG_STATISTICS FALSE`
 - Debug option, kernel statistics.*
- `#define CH_DBG_SYSTEM_STATE_CHECK FALSE`
 - Debug option, system state check.*
- `#define CH_DBG_ENABLE_CHECKS FALSE`
 - Debug option, parameters checks.*
- `#define CH_DBG_ENABLE_ASSERTS FALSE`
 - Debug option, consistency checks.*
- `#define CH_DBG_ENABLE_TRACE FALSE`
 - Debug option, trace buffer.*
- `#define CH_DBG_ENABLE_STACK_CHECK FALSE`
 - Debug option, stack checks.*
- `#define CH_DBG_FILL_THREADS FALSE`
 - Debug option, stacks initialization.*
- `#define CH_DBG_THREADS_PROFILING FALSE`
 - Debug option, threads profiling.*

Kernel hooks

- `#define CH_CFG_THREAD_EXTRA_FIELDS /* Add threads custom fields here.*/`
 - Threads descriptor structure extension.*
- `#define CH_CFG_THREAD_INIT_HOOK(tp)`
 - Threads initialization hook.*
- `#define CH_CFG_THREAD_EXIT_HOOK(tp)`
 - Threads finalization hook.*
- `#define CH_CFG_CONTEXT_SWITCH_HOOK(ntp, otp)`
 - Context switch hook.*
- `#define CH_CFG_IDLE_ENTER_HOOK()`

```
Idle thread enter hook.  
• #define CH_CFG_IDLE_LEAVE_HOOK()  
  
Idle thread leave hook.  
• #define CH_CFG_IDLE_LOOP_HOOK()  
  
Idle Loop hook.  
• #define CH_CFG_SYSTEM_TICK_HOOK()  
  
System tick event hook.  
• #define CH_CFG_SYSTEM_HALT_HOOK(reason)  
  
System halt hook.
```

9.3.2 Macro Definition Documentation

9.3.2.1 #define CH_CFG_ST_RESOLUTION 32

System time counter resolution.

Note

Allowed values are 16 or 32 bits.

9.3.2.2 #define CH_CFG_ST_FREQUENCY 10000

System tick frequency.

Frequency of the system timer that drives the system ticks. This setting also defines the system tick time unit.

9.3.2.3 #define CH_CFG_ST_TIMEDELTA 2

Time delta constant for the tick-less mode.

Note

If this value is zero then the system uses the classic periodic tick. This value represents the minimum number of ticks that is safe to specify in a timeout directive. The value one is not valid, timeouts are rounded up to this value.

9.3.2.4 #define CH_CFG_TIME_QUANTUM 0

Round robin interval.

This constant is the number of system ticks allowed for the threads before preemption occurs. Setting this value to zero disables the preemption for threads with equal priority and the round robin becomes cooperative. Note that higher priority threads can still preempt, the kernel is always preemptive.

Note

Disabling the round robin preemption makes the kernel more compact and generally faster.

The round robin preemption is not supported in tickless mode and must be set to zero in that case.

9.3.2.5 #define CH_CFG_MEMCORE_SIZE 0

Managed RAM size.

Size of the RAM area to be managed by the OS. If set to zero then the whole available RAM is used. The core memory is made available to the heap allocator and/or can be used directly through the simplified core memory allocator.

Note

In order to let the OS manage the whole RAM the linker script must provide the `heap_base` and `heap_end` symbols.

Requires `CH_CFG_USE_MEMCORE`.

9.3.2.6 #define CH_CFG_NO_IDLE_THREAD FALSE

Idle thread automatic spawn suppression.

When this option is activated the function `chSysInit()` does not spawn the idle thread. The application `main()` function becomes the idle thread and must implement an infinite loop.

9.3.2.7 #define CH_CFG_OPTIMIZE_SPEED TRUE

OS optimization.

If enabled then time efficient rather than space efficient code is used when two possible implementations exist.

Note

This is not related to the compiler optimization options.

The default is `TRUE`.

9.3.2.8 #define CH_CFG_USE_TM TRUE

Time Measurement APIs.

If enabled then the time measurement APIs are included in the kernel.

Note

The default is `TRUE`.

9.3.2.9 #define CH_CFG_USE_REGISTRY TRUE

Threads registry APIs.

If enabled then the registry APIs are included in the kernel.

Note

The default is `TRUE`.

9.3.2.10 #define CH_CFG_USE_WAITEXIT TRUE

Threads synchronization APIs.

If enabled then the `chThdWait()` function is included in the kernel.

Note

The default is TRUE.

9.3.2.11 #define CH_CFG_USE_SEMAPHORES TRUE

Semaphores APIs.

If enabled then the Semaphores APIs are included in the kernel.

Note

The default is TRUE.

9.3.2.12 #define CH_CFG_USE_SEMAPHORES_PRIORITY FALSE

Semaphores queuing mode.

If enabled then the threads are enqueued on semaphores by priority rather than in FIFO order.

Note

The default is FALSE. Enable this if you have special requirements.

Requires CH_CFG_USE_SEMAPHORES.

9.3.2.13 #define CH_CFG_USE_MUTEXES TRUE

Mutexes APIs.

If enabled then the mutexes APIs are included in the kernel.

Note

The default is TRUE.

9.3.2.14 #define CH_CFG_USE_MUTEXES_RECURSIVE FALSE

Enables recursive behavior on mutexes.

Note

Recursive mutexes are heavier and have an increased memory footprint.

The default is FALSE.

9.3.2.15 #define CH_CFG_USE_CONDVARIS TRUE

Conditional Variables APIs.

If enabled then the conditional variables APIs are included in the kernel.

Note

The default is TRUE.

Requires CH_CFG_USE_MUTEXES.

9.3.2.16 #define CH_CFG_USE_CONDVAR_TIMEOUT TRUE

Conditional Variables APIs with timeout.

If enabled then the conditional variables APIs with timeout specification are included in the kernel.

Note

The default is TRUE.

Requires CH_CFG_USE_CONDVAR.

9.3.2.17 #define CH_CFG_USE_EVENTS TRUE

Events Flags APIs.

If enabled then the event flags APIs are included in the kernel.

Note

The default is TRUE.

9.3.2.18 #define CH_CFG_USE_EVENTS_TIMEOUT TRUE

Events Flags APIs with timeout.

If enabled then the events APIs with timeout specification are included in the kernel.

Note

The default is TRUE.

Requires CH_CFG_USE_EVENTS.

9.3.2.19 #define CH_CFG_USE_MESSAGES TRUE

Synchronous Messages APIs.

If enabled then the synchronous messages APIs are included in the kernel.

Note

The default is TRUE.

9.3.2.20 #define CH_CFG_USE_MESSAGES_PRIORITY FALSE

Synchronous Messages queuing mode.

If enabled then messages are served by priority rather than in FIFO order.

Note

The default is FALSE. Enable this if you have special requirements.

Requires CH_CFG_USE_MESSAGES.

9.3.2.21 #define CH_CFG_USE_MAILBOXES TRUE

Mailboxes APIs.

If enabled then the asynchronous messages (mailboxes) APIs are included in the kernel.

Note

The default is TRUE.

Requires CH_CFG_USE_SEMAPHORES.

9.3.2.22 #define CH_CFG_USE_QUEUES TRUE

I/O Queues APIs.

If enabled then the I/O queues APIs are included in the kernel.

Note

The default is TRUE.

9.3.2.23 #define CH_CFG_USE_MEMCORE TRUE

Core Memory Manager APIs.

If enabled then the core memory manager APIs are included in the kernel.

Note

The default is TRUE.

9.3.2.24 #define CH_CFG_USE_HEAP TRUE

Heap Allocator APIs.

If enabled then the memory heap allocator APIs are included in the kernel.

Note

The default is TRUE.

Requires CH_CFG_USE_MEMCORE and either CH_CFG_USE_MUTEXES or CH_CFG_USE_SEMAPHORES.

Mutexes are recommended.

9.3.2.25 #define CH_CFG_USE_MEMPOOLS TRUE

Memory Pools Allocator APIs.

If enabled then the memory pools allocator APIs are included in the kernel.

Note

The default is TRUE.

9.3.2.26 #define CH_CFG_USE_DYNAMIC TRUE

Dynamic Threads APIs.

If enabled then the dynamic threads creation APIs are included in the kernel.

Note

The default is TRUE.

Requires CH_CFG_USE_WAITEXIT.

Requires CH_CFG_USE_HEAP and/or CH_CFG_USE_MEMPOOLS.

9.3.2.27 #define CH_DBG_STATISTICS FALSE

Debug option, kernel statistics.

Note

The default is FALSE.

9.3.2.28 #define CH_DBG_SYSTEM_STATE_CHECK FALSE

Debug option, system state check.

If enabled the correct call protocol for system APIs is checked at runtime.

Note

The default is FALSE.

9.3.2.29 #define CH_DBG_ENABLE_CHECKS FALSE

Debug option, parameters checks.

If enabled then the checks on the API functions input parameters are activated.

Note

The default is FALSE.

9.3.2.30 #define CH_DBG_ENABLE_ASSERTS FALSE

Debug option, consistency checks.

If enabled then all the assertions in the kernel code are activated. This includes consistency checks inside the kernel, runtime anomalies and port-defined checks.

Note

The default is FALSE.

9.3.2.31 #define CH_DBG_ENABLE_TRACE FALSE

Debug option, trace buffer.

If enabled then the context switch circular trace buffer is activated.

Note

The default is FALSE.

9.3.2.32 #define CH_DBG_ENABLE_STACK_CHECK FALSE

Debug option, stack checks.

If enabled then a runtime stack check is performed.

Note

The default is FALSE.

The stack check is performed in a architecture/port dependent way. It may not be implemented or some ports.

The default failure mode is to halt the system with the global `panic_msg` variable set to NULL.

9.3.2.33 #define CH_DBG_FILL_THREADS FALSE

Debug option, stacks initialization.

If enabled then the threads working area is filled with a byte value when a thread is created. This can be useful for the runtime measurement of the used stack.

Note

The default is FALSE.

9.3.2.34 #define CH_DBG_THREADS_PROFILING FALSE

Debug option, threads profiling.

If enabled then a field is added to the `thread_t` structure that counts the system ticks occurred while executing the thread.

Note

The default is FALSE.

This debug option is not currently compatible with the tickless mode.

9.3.2.35 #define CH_CFG_THREAD_EXTRA_FIELDS /* Add threads custom fields here. */

Threads descriptor structure extension.

User fields added to the end of the `thread_t` structure.

9.3.2.36 #define CH_CFG_THREAD_INIT_HOOK(tp)**Value:**

```
{\n    /* Add threads initialization code here. */\}
```

Threads initialization hook.

User initialization code added to the `chThdInit()` API.

Note

It is invoked from within `chThdInit()` and implicitly from all the threads creation APIs.

9.3.2.37 #define CH_CFG_THREAD_EXIT_HOOK(*tp*)

Value:

```
{
    /* Add threads finalization code here.*/ \
}
```

Threads finalization hook.

User finalization code added to the `chThdExit()` API.

Note

It is inserted into lock zone.

It is also invoked when the threads simply return in order to terminate.

9.3.2.38 #define CH_CFG_CONTEXT_SWITCH_HOOK(*ntp*, *otp*)

Value:

```
{
    /* Context switch code here.*/ \
}
```

Context switch hook.

This hook is invoked just before switching between threads.

9.3.2.39 #define CH_CFG_IDLE_ENTER_HOOK()

Value:

```
{
}
```

Idle thread enter hook.

Note

This hook is invoked within a critical zone, no OS functions should be invoked from here.

This macro can be used to activate a power saving mode.

9.3.2.40 #define CH_CFG_IDLE_LEAVE_HOOK()

Value:

```
{
}
```

Idle thread leave hook.

Note

This hook is invoked within a critical zone, no OS functions should be invoked from here.
This macro can be used to deactivate a power saving mode.

9.3.2.41 #define CH_CFG_IDLE_LOOP_HOOK()**Value:**

```
{           \
/* Idle loop code here.*\
}
```

Idle Loop hook.

This hook is continuously invoked by the idle thread loop.

9.3.2.42 #define CH_CFG_SYSTEM_TICK_HOOK()**Value:**

```
{           \
/* System tick event code here.*\
}
```

System tick event hook.

This hook is invoked in the system tick handler immediately after processing the virtual timers queue.

9.3.2.43 #define CH_CFG_SYSTEM_HALT_HOOK(reason)**Value:**

```
{           \
/* System halt code here.*\
}
```

System halt hook.

This hook is invoked in case to a system halting error before the system is halted.

9.4 Kernel Types

9.4.1 Detailed Description

Macros

- `#define ROMCONST const`
ROM constant modifier.
- `#define NOINLINE __attribute__((noinline))`
Makes functions not inlineable.
- `#define PORT_THD_FUNCTION(tname, arg) msg_t tname(void *arg)`
Optimized thread function declaration macro.
- `#define PACKED_VAR __attribute__((packed))`
Packed variable specifier.

Common constants

- `#define FALSE 0`
Generic 'false' boolean constant.
- `#define TRUE (!FALSE)`
Generic 'true' boolean constant.

Kernel types

- `typedef uint32_t rtcnt_t`
- `typedef uint64_t rttime_t`
- `typedef uint32_t syssts_t`
- `typedef uint8_t tmode_t`
- `typedef uint8_t tstate_t`
- `typedef uint8_t trefs_t`
- `typedef uint8_t tslices_t`
- `typedef uint32_t tprio_t`
- `typedef int32_t msg_t`
- `typedef int32_t eventid_t`
- `typedef uint32_t eventmask_t`
- `typedef uint32_t eventflags_t`
- `typedef int32_t cnt_t`
- `typedef uint32_t ucnt_t`

9.4.2 Macro Definition Documentation

9.4.2.1 `#define FALSE 0`

Generic 'false' boolean constant.

9.4.2.2 `#define TRUE (!FALSE)`

Generic 'true' boolean constant.

9.4.2.3 #define ROMCONST const

ROM constant modifier.

Note

It is set to use the "const" keyword in this port.

9.4.2.4 #define NOINLINE __attribute__((noinline))

Makes functions not inlineable.

Note

If the compiler does not support such attribute then the realtime counter precision could be degraded.

9.4.2.5 #define PORT_THD_FUNCTION(*tname*, *arg*) msg_t *tname*(void **arg*)

Optimized thread function declaration macro.

9.4.2.6 #define PACKED_VAR __attribute__((packed))

Packed variable specifier.

9.4.3 Typedef Documentation

9.4.3.1 typedef uint32_t rtcnt_t

Realtime counter.

9.4.3.2 typedef uint64_t rttime_t

Realtime accumulator.

9.4.3.3 typedef uint32_t syssts_t

System status word.

9.4.3.4 typedef uint8_t tmode_t

Thread flags.

9.4.3.5 typedef uint8_t tstate_t

Thread state.

9.4.3.6 typedef uint8_t trefs_t

Thread references counter.

9.4.3.7 `typedef uint8_t tslices_t`

Thread time slices counter.

9.4.3.8 `typedef uint32_t tprio_t`

Thread priority.

9.4.3.9 `typedef int32_t msg_t`

Inter-thread message.

9.4.3.10 `typedef int32_t eventid_t`

Numeric event identifier.

9.4.3.11 `typedef uint32_t eventmask_t`

Mask of event identifiers.

9.4.3.12 `typedef uint32_t eventflags_t`

Mask of event flags.

9.4.3.13 `typedef int32_t cnt_t`

Generic signed counter.

9.4.3.14 `typedef uint32_t ucnt_t`

Generic unsigned counter.

9.5 Base Kernel Services

9.5.1 Detailed Description

Base kernel services, the base subsystems are always included in the OS builds.

Modules

- [System Management](#)
- [Scheduler](#)
- [Threads](#)
- [Time and Virtual Timers](#)

9.6 System Management

9.6.1 Detailed Description

System related APIs and services:

- Initialization.
- Locks.
- Interrupt Handling.
- Power Management.
- Abnormal Termination.
- Realtime counter.

Macros

- `#define chSysGetRealtimeCounterX() (rtcnt_t)port_rt_get_counter_value()`
Returns the current value of the system real time counter.
- `#define chSysSwitch(ntp, otp)`
Performs a context switch.

Masks of executable integrity checks.

- `#define CH_INTEGRITY_RLIST 1U`
- `#define CH_INTEGRITY_VTLIST 2U`
- `#define CH_INTEGRITY_REGISTRY 4U`
- `#define CH_INTEGRITY_PORT 8U`

ISRs abstraction macros

- `#define CH_IRQ_IS_VALID_PRIORITY(prio) PORT_IRQ_IS_VALID_PRIORITY(prio)`
Priority level validation macro.
- `#define CH_IRQ_IS_VALID_KERNEL_PRIORITY(prio) PORT_IRQ_IS_VALID_KERNEL_PRIORITY(prio)`
Priority level validation macro.
- `#define CH_IRQ_PROLOGUE()`
IRQ handler enter code.
- `#define CH_IRQ_EPILOGUE()`
IRQ handler exit code.
- `#define CH_IRQ_HANDLER(id) PORT_IRQ_HANDLER(id)`
Standard normal IRQ handler declaration.

Fast ISRs abstraction macros

- `#define CH_FAST_IRQ_HANDLER(id) PORT_FAST_IRQ_HANDLER(id)`
Standard fast IRQ handler declaration.

Time conversion utilities for the realtime counter

- `#define S2RTC(freq, sec) ((freq) * (sec))`
Seconds to realtime counter.
- `#define MS2RTC(freq, msec) (rtcnt_t)((((freq) + 999UL) / 1000UL) * (msec))`
Milliseconds to realtime counter.
- `#define US2RTC(freq, usec) (rtcnt_t)((((freq) + 999999UL) / 1000000UL) * (usec))`
Microseconds to realtime counter.
- `#define RTC2S(freq, n) (((n) - 1UL) / (freq)) + 1UL`
Realtime counter cycles to seconds.
- `#define RTC2MS(freq, n) (((n) - 1UL) / ((freq) / 1000UL)) + 1UL`
Realtime counter cycles to milliseconds.
- `#define RTC2US(freq, n) (((n) - 1UL) / ((freq) / 1000000UL)) + 1UL`
Realtime counter cycles to microseconds.

Functions

- `static void _idle_thread (void *p)`
This function implements the idle thread infinite loop.
- `void chSysInit (void)`
ChibiOS/RT initialization.
- `void chSysHalt (const char *reason)`
Halts the system.
- `bool chSysIntegrityCheck (unsigned testmask)`
System integrity check.
- `void chSysTimerHandler (void)`
Handles time ticks for round robin preemption and timer increments.
- `syssts_t chSysGetStatusAndLockX (void)`
Returns the execution status and enters a critical zone.
- `void chSysRestoreStatusX (syssts_t sts)`
Restores the specified execution status and leaves a critical zone.
- `bool chSysIsCounterWithinX (rtcnt_t cnt, rtcnt_t start, rtcnt_t end)`
Realtime window test.
- `void chSysPolledDelayX (rtcnt_t cycles)`
Polled delay.
- `static void chSysDisable (void)`
Raises the system interrupt priority mask to the maximum level.
- `static void chSysSuspend (void)`
Raises the system interrupt priority mask to system level.
- `static void chSysEnable (void)`
Lowers the system interrupt priority mask to user level.
- `static void chSysLock (void)`
Enters the kernel lock state.
- `static void chSysUnlock (void)`
Leaves the kernel lock state.
- `static void chSysLockFromISR (void)`
Enters the kernel lock state from within an interrupt handler.
- `static void chSysUnlockFromISR (void)`
Leaves the kernel lock state from within an interrupt handler.
- `static void chSysUnconditionalLock (void)`
Unconditionally enters the kernel lock state.

- static void `chSysUnconditionalUnlock` (void)
Unconditionally leaves the kernel lock state.
- static `thread_t * chSysGetIdleThreadX` (void)
Returns a pointer to the idle thread.

9.6.2 Macro Definition Documentation

9.6.2.1 #define CH_IRQ_IS_VALID_PRIORITY(*prio*) PORT_IRQ_IS_VALID_PRIORITY(*prio*)

Priority level validation macro.

This macro determines if the passed value is a valid priority level for the underlying architecture.

Parameters

in	<i>prio</i>	the priority level
----	-------------	--------------------

Returns

Priority range result.

Return values

<i>false</i>	if the priority is invalid or if the architecture does not support priorities.
<i>true</i>	if the priority is valid.

9.6.2.2 #define CH_IRQ_IS_VALID_KERNEL_PRIORITY(*prio*) PORT_IRQ_IS_VALID_KERNEL_PRIORITY(*prio*)

Priority level validation macro.

This macro determines if the passed value is a valid priority level that cannot preempt the kernel critical zone.

Parameters

in	<i>prio</i>	the priority level
----	-------------	--------------------

Returns

Priority range result.

Return values

<i>false</i>	if the priority is invalid or if the architecture does not support priorities.
<i>true</i>	if the priority is valid.

9.6.2.3 #define CH_IRQ_PROLOGUE()

Value:

```
PORT_IRQ_PROLOGUE();
  \
  _stats_increase_irq();
  \
  _dbg_check_enter_isr()
```

IRQ handler enter code.

Note

Usually IRQ handlers functions are also declared naked.
On some architectures this macro can be empty.

Function Class:

Special function, this function has special requirements see the notes.

9.6.2.4 #define CH_IRQ_EPILOGUE()**Value:**

```
_dbg_check_leave_isr();  
PORT_IRQ_EPILOGUE()
```

IRQ handler exit code.

Note

Usually IRQ handlers function are also declared naked.
This macro usually performs the final reschedule by using `chSchIsPreemptionRequired()` and `chSchDoReschedule()`.

Function Class:

Special function, this function has special requirements see the notes.

9.6.2.5 #define CH_IRQ_HANDLER(id) PORT_IRQ_HANDLER(id)

Standard normal IRQ handler declaration.

Note

`id` can be a function name or a vector number depending on the port implementation.

Function Class:

Special function, this function has special requirements see the notes.

9.6.2.6 #define CH_FAST_IRQ_HANDLER(id) PORT_FAST_IRQ_HANDLER(id)

Standard fast IRQ handler declaration.

Note

`id` can be a function name or a vector number depending on the port implementation.
Not all architectures support fast interrupts.

Function Class:

Special function, this function has special requirements see the notes.

9.6.2.7 #define S2RTC(freq, sec) ((freq) * (sec))

Seconds to realtime counter.

Converts from seconds to realtime counter cycles.

Note

The macro assumes that `freq >= 1`.

Parameters

in	<i>freq</i>	clock frequency, in Hz, of the realtime counter
in	<i>sec</i>	number of seconds

Returns

The number of cycles.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

9.6.2.8 #define MS2RTC(freq, msec) (rtcnt_t)((((freq) + 999UL) / 1000UL) * (msec))

Milliseconds to realtime counter.

Converts from milliseconds to realtime counter cycles.

Note

The result is rounded upward to the next millisecond boundary.

The macro assumes that `freq >= 1000`.

Parameters

in	<i>freq</i>	clock frequency, in Hz, of the realtime counter
in	<i>msec</i>	number of milliseconds

Returns

The number of cycles.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

9.6.2.9 #define US2RTC(freq, usec) (rtcnt_t)((((freq) + 999999UL) / 1000000UL) * (usec))

Microseconds to realtime counter.

Converts from microseconds to realtime counter cycles.

Note

The result is rounded upward to the next microsecond boundary.

The macro assumes that `freq >= 1000000`.

Parameters

in	<i>freq</i>	clock frequency, in Hz, of the realtime counter
in	<i>usec</i>	number of microseconds

Returns

The number of cycles.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

9.6.2.10 #define RTC2S(*freq*, *n*) (((n) - 1UL) / (*freq*) + 1UL)

Realtime counter cycles to seconds.

Converts from realtime counter cycles number to seconds.

Note

The result is rounded up to the next second boundary.

The macro assumes that *freq* ≥ 1 .

Parameters

in	<i>freq</i>	clock frequency, in Hz, of the realtime counter
in	<i>n</i>	number of cycles

Returns

The number of seconds.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

9.6.2.11 #define RTC2MS(*freq*, *n*) (((n) - 1UL) / ((*freq*) / 1000UL)) + 1UL

Realtime counter cycles to milliseconds.

Converts from realtime counter cycles number to milliseconds.

Note

The result is rounded up to the next millisecond boundary.

The macro assumes that *freq* ≥ 1000 .

Parameters

in	<i>freq</i>	clock frequency, in Hz, of the realtime counter
in	<i>n</i>	number of cycles

Returns

The number of milliseconds.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

9.6.2.12 #define RTC2US(freq, n) (((n) - 1UL) / ((freq) / 1000000UL)) + 1UL)

Realtime counter cycles to microseconds.

Converts from realtime counter cycles number to microseconds.

Note

The result is rounded up to the next microsecond boundary.
The macro assumes that freq >= 1000000.

Parameters

in	<i>freq</i>	clock frequency, in Hz, of the realtime counter
in	<i>n</i>	number of cycles

Returns

The number of microseconds.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

9.6.2.13 #define chSysGetRealtimeCounterX() (rtcnt_t)port_rt_get_counter_value()

Returns the current value of the system real time counter.

Note

This function is only available if the port layer supports the option PORT_SUPPORTS_RT.

Returns

The value of the system realtime counter of type rtcnt_t.

Function Class:

This is an **X-Class** API, this function can be invoked from any context.

9.6.2.14 #define chSysSwitch(ntp, otp)

Value:

```
{
    \_dbg_trace(otp);
    \_stats_ctxswc(ntp, otp);
    \CH_CFG_CONTEXT_SWITCH_HOOK(ntp, otp);
    \port_switch(ntp, otp);
}
```

Performs a context switch.

Note

Not a user function, it is meant to be invoked by the scheduler itself or from within the port layer.

Parameters

in	<i>ntp</i>	the thread to be switched in
in	<i>otp</i>	the thread to be switched out

Function Class:

Special function, this function has special requirements see the notes.

9.6.3 Function Documentation**9.6.3.1 static void _idle_thread (void * *p*) [static]**

This function implements the idle thread infinite loop.

The function puts the processor in the lowest power mode capable to serve interrupts.

The priority is internally set to the minimum system value so that this thread is executed only if there are no other ready threads in the system.

Parameters

in	<i>p</i>	the thread parameter, unused in this scenario
----	----------	---

Here is the call graph for this function:

**9.6.3.2 void chSysInit (void)**

ChibiOS/RT initialization.

After executing this function the current instructions stream becomes the main thread.

Precondition

Interrupts must be disabled before invoking this function.

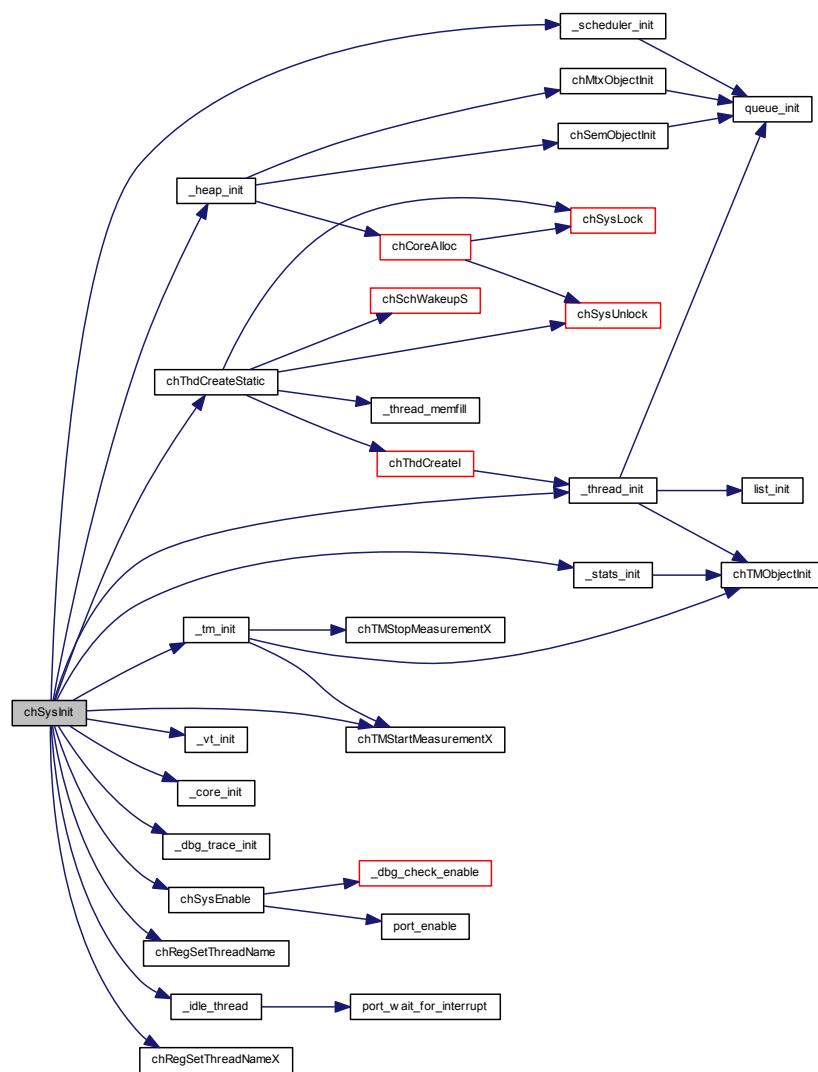
Postcondition

The main thread is created with priority NORMALPRIO and interrupts are enabled.

Function Class:

Special function, this function has special requirements see the notes.

Here is the call graph for this function:



9.6.3.3 void chSysHalt(const char * reason)

Halts the system.

This function is invoked by the operating system when an unrecoverable error is detected, for example because a programming error in the application code that triggers an assertion while in debug mode.

Note

Can be invoked from any system state.

Parameters

in	<i>reason</i>	pointer to an error string
----	---------------	----------------------------

Function Class:

Special function, this function has special requirements see the notes.

Here is the call graph for this function:

**9.6.3.4 bool chSysIntegrityCheck(unsigned *testmask*)**

System integrity check.

Performs an integrity check of the important ChibiOS/RT data structures.

Note

The appropriate action in case of failure is to halt the system before releasing the critical zone.

If the system is corrupted then one possible outcome of this function is an exception caused by NULL or corrupted pointers in list elements. Exception vectors must be monitored as well.

This function is not used internally, it is up to the application to define if and where to perform system checking. Performing all tests at once can be a slow operation and can degrade the system response time. It is suggested to execute one test at time and release the critical zone in between tests.

Parameters

in	<i>testmask</i>	Each bit in this mask is associated to a test to be performed.
----	-----------------	--

Returns

The test result.

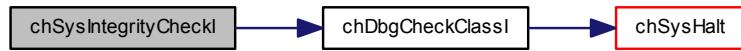
Return values

<i>false</i>	The test succeeded.
<i>true</i>	Test failed.

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:



9.6.3.5 void chSysTimerHandler() (void)

Handles time ticks for round robin preemption and timer increments.

Decrements the remaining time quantum of the running thread and preempts it when the quantum is used up.
Increments system time and manages the timers.

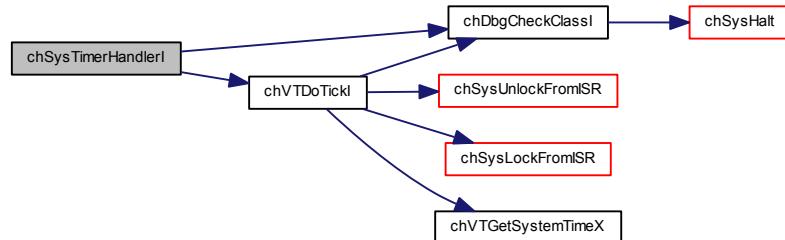
Note

The frequency of the timer determines the system tick granularity and, together with the CH_CFG_TIME_QUANTUM macro, the round robin interval.

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:



9.6.3.6 syssts_t chSysGetStatusAndLockX() (void)

Returns the execution status and enters a critical zone.

This functions enters into a critical zone and can be called from any context. Because its flexibility it is less efficient than [chSysLock\(\)](#) which is preferable when the calling context is known.

Postcondition

The system is in a critical zone.

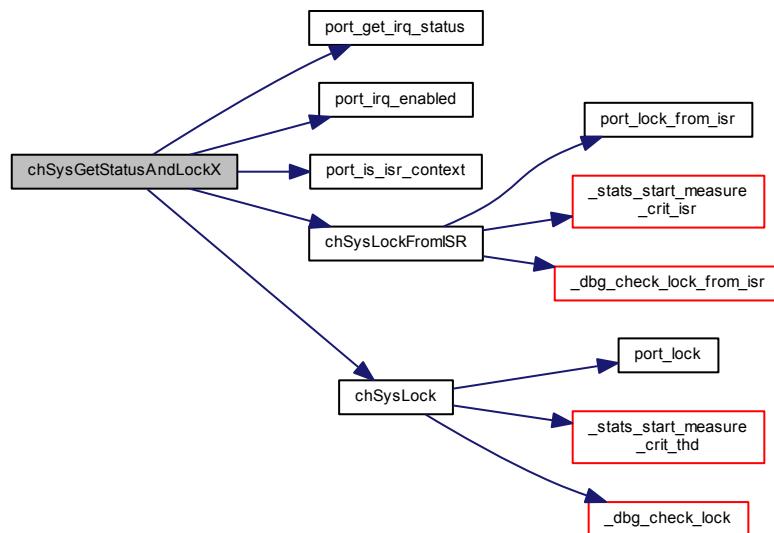
Returns

The previous system status, the encoding of this status word is architecture-dependent and opaque.

Function Class:

This is an **X-Class** API, this function can be invoked from any context.

Here is the call graph for this function:

**9.6.3.7 void chSysRestoreStatusX (syssts_t sts)**

Restores the specified execution status and leaves a critical zone.

Note

A call to `chSchRescheduleS()` is automatically performed if exiting the critical zone and if not in ISR context.

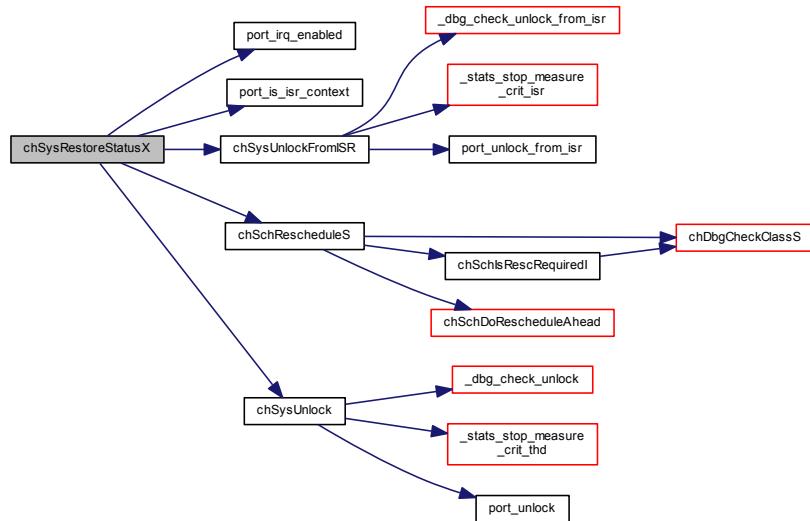
Parameters

in	<i>sts</i>	the system status to be restored.
----	------------	-----------------------------------

Function Class:

This is an **X-Class** API, this function can be invoked from any context.

Here is the call graph for this function:

**9.6.3.8 bool chSysIsCounterWithinX(rtcnt_t cnt, rtcnt_t start, rtcnt_t end)**

Realtime window test.

This function verifies if the current realtime counter value lies within the specified range or not. The test takes care of the realtime counter wrapping to zero on overflow.

Note

When `start==end` then the function returns always true because the whole time range is specified.
This function is only available if the port layer supports the option `PORT_SUPPORTS_RT`.

Parameters

in	<i>cnt</i>	the counter value to be tested
in	<i>start</i>	the start of the time window (inclusive)
in	<i>end</i>	the end of the time window (non inclusive)

Return values

<i>true</i>	current time within the specified time window.
<i>false</i>	current time not within the specified time window.

Function Class:

This is an **X-Class** API, this function can be invoked from any context.

9.6.3.9 void chSysPolledDelayX (*rtcnt_t cycles*)

Polled delay.

Note

The real delay is always few cycles in excess of the specified value.

This function is only available if the port layer supports the option `PORT_SUPPORTS_RT`.

Parameters

<i>in</i>	<i>cycles</i>	number of cycles
-----------	---------------	------------------

Function Class:

This is an **X-Class** API, this function can be invoked from any context.

Here is the call graph for this function:

9.6.3.10 static void chSysDisable (*void*) [inline], [static]

Raises the system interrupt priority mask to the maximum level.

All the maskable interrupt sources are disabled regardless their hardware priority.

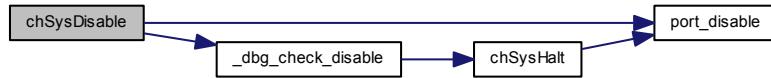
Note

Do not invoke this API from within a kernel lock.

Function Class:

Special function, this function has special requirements see the notes.

Here is the call graph for this function:

**9.6.3.11 static void chSysSuspend(void) [inline], [static]**

Raises the system interrupt priority mask to system level.

The interrupt sources that should not be able to preempt the kernel are disabled, interrupt sources with higher priority are still enabled.

Note

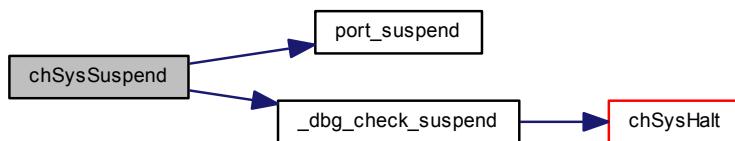
Do not invoke this API from within a kernel lock.

This API is no replacement for [chSysLock\(\)](#), the [chSysLock\(\)](#) could do more than just disable the interrupts.

Function Class:

Special function, this function has special requirements see the notes.

Here is the call graph for this function:

**9.6.3.12 static void chSysEnable(void) [inline], [static]**

Lowers the system interrupt priority mask to user level.

All the interrupt sources are enabled.

Note

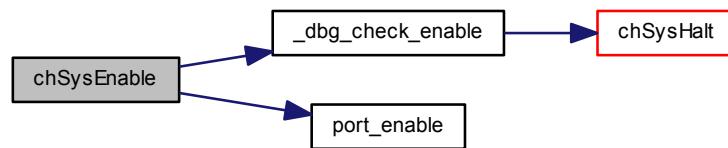
Do not invoke this API from within a kernel lock.

This API is no replacement for `chSysUnlock()`, the `chSysUnlock()` could do more than just enable the interrupts.

Function Class:

Special function, this function has special requirements see the notes.

Here is the call graph for this function:

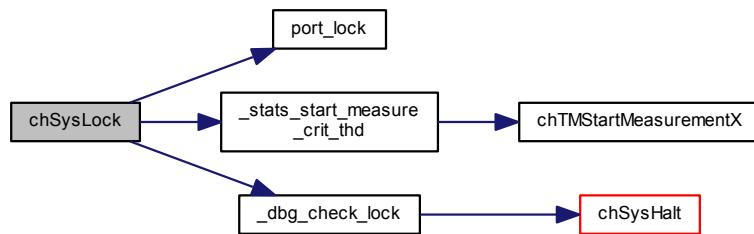
**9.6.3.13 static void chSysLock (void) [inline], [static]**

Enters the kernel lock state.

Function Class:

Special function, this function has special requirements see the notes.

Here is the call graph for this function:

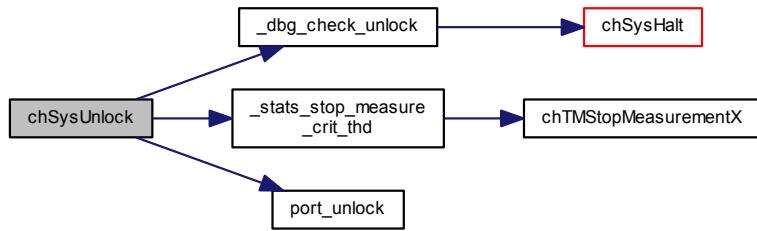
**9.6.3.14 static void chSysUnlock (void) [inline], [static]**

Leaves the kernel lock state.

Function Class:

Special function, this function has special requirements see the notes.

Here is the call graph for this function:



9.6.3.15 static void chSysLockFromISR (void) [inline], [static]

Enters the kernel lock state from within an interrupt handler.

Note

This API may do nothing on some architectures, it is required because on ports that support preemptable interrupt handlers it is required to raise the interrupt mask to the same level of the system mutual exclusion zone.

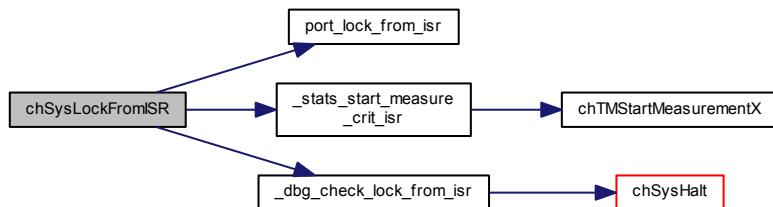
It is good practice to invoke this API before invoking any I-class syscall from an interrupt handler.

This API must be invoked exclusively from interrupt handlers.

Function Class:

Special function, this function has special requirements see the notes.

Here is the call graph for this function:



9.6.3.16 static void chSysUnlockFromISR (void) [inline], [static]

Leaves the kernel lock state from within an interrupt handler.

Note

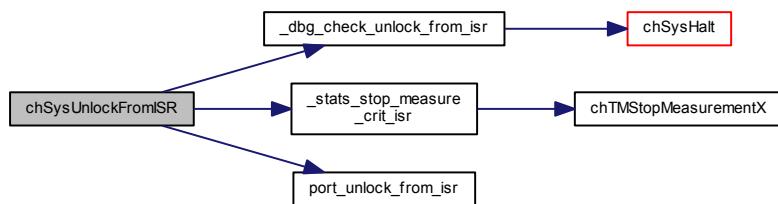
This API may do nothing on some architectures, it is required because on ports that support preemptable interrupt handlers it is required to raise the interrupt mask to the same level of the system mutual exclusion zone.

It is good practice to invoke this API after invoking any I-class syscall from an interrupt handler.
This API must be invoked exclusively from interrupt handlers.

Function Class:

Special function, this function has special requirements see the notes.

Here is the call graph for this function:

**9.6.3.17 static void chSysUnconditionalLock (void) [inline], [static]**

Unconditionally enters the kernel lock state.

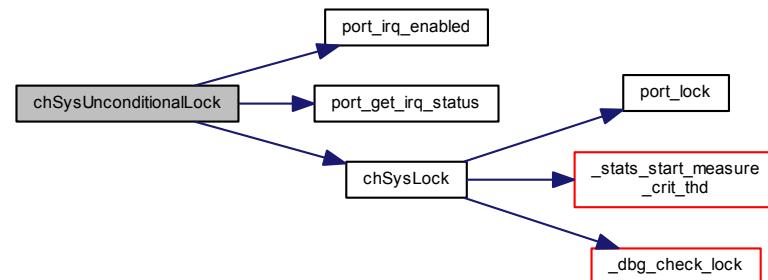
Note

Can be called without previous knowledge of the current lock state. The final state is "s-locked".

Function Class:

Special function, this function has special requirements see the notes.

Here is the call graph for this function:



9.6.3.18 static void chSysUnconditionalUnlock(void) [inline], [static]

Unconditionally leaves the kernel lock state.

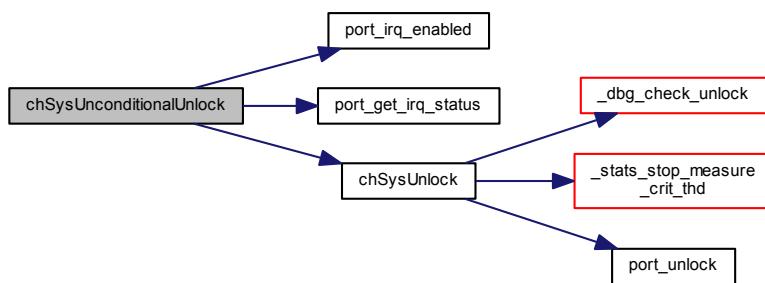
Note

Can be called without previous knowledge of the current lock state. The final state is "normal".

Function Class:

Special function, this function has special requirements see the notes.

Here is the call graph for this function:



9.6.3.19 static thread_t* chSysGetIdleThreadX(void) [inline], [static]

Returns a pointer to the idle thread.

Precondition

In order to use this function the option CH_CFG_NO_IDLE_THREAD must be disabled.

Note

The reference counter of the idle thread is not incremented but it is not strictly required being the idle thread a static object.

Returns

Pointer to the idle thread.

Function Class:

This is an **X-Class** API, this function can be invoked from any context.

9.7 Scheduler

9.7.1 Detailed Description

This module provides the default portable scheduler code.

Macros

- `#define firstprio(rlp) ((rlp)->p_next->p_prio)`
Returns the priority of the first thread on the given ready list.
- `#define currp ch.rlist.r_current`
Current thread pointer access macro.
- `#define setcurrp(tp) (currp = (tp))`
Current thread pointer change macro.

Wakeup status codes

- `#define MSG_OK (msg_t)0`
Normal wakeup message.
- `#define MSG_TIMEOUT (msg_t)-1`
Wakeup caused by a timeout condition.
- `#define MSG_RESET (msg_t)-2`
Wakeup caused by a reset condition.

Priority constants

- `#define NOPRIO (tprio_t)0`
Ready list header priority.
- `#define IDLEPRIO (tprio_t)1`
Idle priority.
- `#define LOWPRIO (tprio_t)2`
Lowest priority.
- `#define NORMALPRIO (tprio_t)64`
Normal priority.
- `#define HIGHPRIO (tprio_t)127`
Highest priority.
- `#define ABSPRIO (tprio_t)255`
Greatest priority.

Thread states

- `#define CH_STATE_READY (tstate_t)0`
Waiting on the ready list.
- `#define CH_STATE_CURRENT (tstate_t)1`
Currently running.
- `#define CH_STATE_WTSTART (tstate_t)2`
Just created.
- `#define CH_STATE_SUSPENDED (tstate_t)3`
Suspended state.
- `#define CH_STATE_QUEUED (tstate_t)4`

- `#define CH_STATE_WTSEM (tstate_t)5`
On an I/O queue.
- `#define CH_STATE_WTMTX (tstate_t)6`
On a semaphore.
- `#define CH_STATE_WTCOND (tstate_t)7`
On a mutex.
- `#define CH_STATE_SLEEPING (tstate_t)8`
On a cond.variable.
- `#define CH_STATE_WTEXIT (tstate_t)9`
Waiting a thread.
- `#define CH_STATE_WTOREVT (tstate_t)10`
One event.
- `#define CH_STATE_WTANDEVT (tstate_t)11`
Several events.
- `#define CH_STATE SNDMSGQ (tstate_t)12`
Sending a message, in queue.
- `#define CH_STATE SNDMSG (tstate_t)13`
Sent a message, waiting answer.
- `#define CH_STATE_WTMSG (tstate_t)14`
Waiting for a message.
- `#define CH_STATE_FINAL (tstate_t)15`
Thread terminated.
- `#define CH_STATE_NAMES`
Thread states as array of strings.

Thread flags and attributes

- `#define CH_FLAG_MODE_MASK (tmode_t)3U`
Thread memory mode mask.
- `#define CH_FLAG_MODE_STATIC (tmode_t)0U`
Static thread.
- `#define CH_FLAG_MODE_HEAP (tmode_t)1U`
Thread allocated from a Memory Heap.
- `#define CH_FLAG_MODE_MPOOL (tmode_t)2U`
Thread allocated from a Memory Pool.
- `#define CH_FLAG_TERMINATE (tmode_t)4U`
Termination requested flag.

Working Areas and Alignment

- `#define THD_ALIGN_STACK_SIZE(n) (((((size_t)(n)) - 1U) | (sizeof(stkalign_t) - 1U)) + 1U)`
Enforces a correct alignment for a stack area size value.
- `#define THD_WORKING_AREA_SIZE(n) THD_ALIGN_STACK_SIZE(sizeof(thread_t) + PORT_WA_SIZE(n))`
Calculates the total Working Area size.
- `#define THD_WORKING_AREA(s, n) stkalign_t s[THD_WORKING_AREA_SIZE(n) / sizeof(stkalign_t)]`
Static working area allocation.

Threads abstraction macros

- `#define THD_FUNCTION(tname, arg) PORT_THD_FUNCTION(tname, arg)`
Thread declaration macro.

Typedefs

- `typedef uint32_t systime_t`
Type of system time.
- `typedef struct ch_thread thread_t`
Type of a thread structure.
- `typedef thread_t * thread_reference_t`
Type of a thread reference.
- `typedef struct ch_threads_list threads_list_t`
Type of a generic threads single link list, it works like a stack.
- `typedef struct ch_threads_queue threads_queue_t`
Type of a generic threads bidirectional linked list header and element.
- `typedef struct ch_ready_list ready_list_t`
Type of a ready list header.
- `typedef void(* vfunc_t) (void *p)`
Type of a Virtual Timer callback function.
- `typedef struct ch_virtual_timer virtual_timer_t`
Type of a Virtual Timer structure.
- `typedef struct ch_virtual_timers_list virtual_timers_list_t`
Type of virtual timers list header.
- `typedef struct ch_system_debug system_debug_t`
Type of a system debug structure.
- `typedef struct ch_system ch_system_t`
Type of system data structure.

Data Structures

- `struct ch_threads_list`
Generic threads single link list, it works like a stack.
- `struct ch_threads_queue`
Generic threads bidirectional linked list header and element.
- `struct ch_thread`
Structure representing a thread.
- `struct ch_virtual_timer`
Virtual Timer descriptor structure.
- `struct ch_virtual_timers_list`
Virtual timers list header.
- `struct ch_system_debug`
System debug data structure.
- `struct ch_system`
System data structure.

Functions

- void `_scheduler_init` (void)

Scheduler initialization.
- void `queue_prio_insert` (`thread_t` *tp, `threads_queue_t` *tqp)

Inserts a thread into a priority ordered queue.
- void `queue_insert` (`thread_t` *tp, `threads_queue_t` *tqp)

Inserts a thread into a queue.
- `thread_t` * `queue_fifo_remove` (`threads_queue_t` *tqp)

Removes the first-out thread from a queue and returns it.
- `thread_t` * `queue_lifo_remove` (`threads_queue_t` *tqp)

Removes the last-out thread from a queue and returns it.
- `thread_t` * `queue_dequeue` (`thread_t` *tp)

Removes a thread from a queue and returns it.
- void `list_insert` (`thread_t` *tp, `threads_list_t` *tlp)

Pushes a thread_t on top of a stack list.
- `thread_t` * `list_remove` (`threads_list_t` *tlp)

Pops a thread from the top of a stack list and returns it.
- `thread_t` * `chSchReadyL` (`thread_t` *tp)

Inserts a thread in the Ready List.
- void `chSchGoSleepS` (`tstate_t` newstate)

Puts the current thread to sleep into the specified state.
- `msg_t` `chSchGoSleepTimeoutS` (`tstate_t` newstate, `systime_t` time)

Puts the current thread to sleep into the specified state with timeout specification.
- void `chSchWakeupS` (`thread_t` *ntp, `msg_t` msg)

Wakes up a thread.
- void `chSchRescheduleS` (void)

Performs a reschedule if a higher priority thread is runnable.
- bool `chSchIsPreemptionRequired` (void)

Evaluates if preemption is required.
- void `chSchDoRescheduleBehind` (void)

Switches to the first thread on the runnable queue.
- void `chSchDoRescheduleAhead` (void)

Switches to the first thread on the runnable queue.
- void `chSchDoReschedule` (void)

Switches to the first thread on the runnable queue.
- static void `list_init` (`threads_list_t` *tlp)

Threads list initialization.
- static bool `list_isempty` (`threads_list_t` *tlp)

Evaluates to true if the specified threads list is empty.
- static bool `list_notempty` (`threads_list_t` *tlp)

Evaluates to true if the specified threads list is not empty.
- static void `queue_init` (`threads_queue_t` *tqp)

Threads queue initialization.
- static bool `queue_isempty` (const `threads_queue_t` *tqp)

Evaluates to true if the specified threads queue is empty.
- static bool `queue_notempty` (const `threads_queue_t` *tqp)

Evaluates to true if the specified threads queue is not empty.
- static bool `chSchIsRescRequired` (void)

Determines if the current thread must reschedule.
- static bool `chSchCanYieldS` (void)

- Determines if yielding is possible.*
- static void `chSchDoYieldS` (void)
Yields the time slot.
 - static void `chSchPreemption` (void)
Inline-able preemption code.

Variables

- `ch_system_t ch`
System data structures.

9.7.2 Macro Definition Documentation

9.7.2.1 #define MSG_OK (msg_t)0

Normal wakeup message.

9.7.2.2 #define MSG_TIMEOUT (msg_t)-1

Wakeup caused by a timeout condition.

9.7.2.3 #define MSG_RESET (msg_t)-2

Wakeup caused by a reset condition.

9.7.2.4 #define NOPRIO (tprio_t)0

Ready list header priority.

9.7.2.5 #define IDLEPRIO (tprio_t)1

Idle priority.

9.7.2.6 #define LOWPRIO (tprio_t)2

Lowest priority.

9.7.2.7 #define NORMALPRIO (tprio_t)64

Normal priority.

9.7.2.8 #define HIGHPRIO (tprio_t)127

Highest priority.

9.7.2.9 #define ABSPRIO (tprio_t)255

Greatest priority.

9.7.2.10 #define CH_STATE_READY (tstate_t)0

Waiting on the ready list.

9.7.2.11 #define CH_STATE_CURRENT (tstate_t)1

Currently running.

9.7.2.12 #define CH_STATE_WTSTART (tstate_t)2

Just created.

9.7.2.13 #define CH_STATE_SUSPENDED (tstate_t)3

Suspended state.

9.7.2.14 #define CH_STATE_QUEUED (tstate_t)4

On an I/O queue.

9.7.2.15 #define CH_STATE_WTSEM (tstate_t)5

On a semaphore.

9.7.2.16 #define CH_STATE_WTMutex (tstate_t)6

On a mutex.

9.7.2.17 #define CH_STATE_WTCOND (tstate_t)7

On a cond.variable.

9.7.2.18 #define CH_STATE_SLEEPING (tstate_t)8

Sleeping.

9.7.2.19 #define CH_STATE_WTEXIT (tstate_t)9

Waiting a thread.

9.7.2.20 #define CH_STATE_WTOREVT (tstate_t)10

One event.

9.7.2.21 #define CH_STATE_WTANDEVT (tstate_t)11

Several events.

9.7.2.22 #define CH_STATE SNDMSGQ (tstate_t)12

Sending a message, in queue.

9.7.2.23 #define CH_STATE SNDMSG (tstate_t)13

Sent a message, waiting answer.

9.7.2.24 #define CH_STATE_WTMSG (tstate_t)14

Waiting for a message.

9.7.2.25 #define CH_STATE_FINAL (tstate_t)15

Thread terminated.

9.7.2.26 #define CH_STATE_NAMES

Value:

```
"READY", "CURRENT", "WTSTART", "SUSPENDED", "QUEUED", "WTSEM", "WTMTX",  \
"WTCOND", "SLEEPING", "WTEXIT", "WTOREVT", "WTANDEV", "SNDMSGQ", \
"SNDMSG", "WTMSG", "FINAL"
```

Thread states as array of strings.

Each element in an array initialized with this macro can be indexed using the numeric thread state values.

9.7.2.27 #define CH_FLAG_MODE_MASK (tmode_t)3U

Thread memory mode mask.

9.7.2.28 #define CH_FLAG_MODE_STATIC (tmode_t)0U

Static thread.

9.7.2.29 #define CH_FLAG_MODE_HEAP (tmode_t)1U

Thread allocated from a Memory Heap.

9.7.2.30 #define CH_FLAG_MODE_MPOOL (tmode_t)2U

Thread allocated from a Memory Pool.

9.7.2.31 #define CH_FLAG_TERMINATE (tmode_t)4U

Termination requested flag.

9.7.2.32 #define THD_ALIGN_STACK_SIZE(n) (((size_t)(n)) - 1U) | (sizeof(stkalign_t) - 1U)) + 1U)

Enforces a correct alignment for a stack area size value.

Parameters

in	<i>n</i>	the stack size to be aligned to the next stack alignment boundary
----	----------	---

Returns

The aligned stack size.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

9.7.2.33 #define THD_WORKING_AREA_SIZE(*n*) THD_ALIGN_STACK_SIZE(sizeof(thread_t) + PORT_WA_SIZE(*n*))

Calculates the total Working Area size.

Parameters

in	<i>n</i>	the stack size to be assigned to the thread
----	----------	---

Returns

The total used memory in bytes.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

9.7.2.34 #define THD_WORKING_AREA(*s*, *n*) stkalign_t s[THD_WORKING_AREA_SIZE(*n*) / sizeof(stkalign_t)]

Static working area allocation.

This macro is used to allocate a static thread working area aligned as both position and size.

Parameters

in	<i>s</i>	the name to be assigned to the stack array
in	<i>n</i>	the stack size to be assigned to the thread

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

9.7.2.35 #define THD_FUNCTION(*tname*, *arg*) PORT_THD_FUNCTION(*tname*, *arg*)

Thread declaration macro.

Note

Thread declarations should be performed using this macro because the port layer could define optimizations for thread functions.

9.7.2.36 #define firstprio(*rlp*) ((rlp)->p_next->p_prio)

Returns the priority of the first thread on the given ready list.

Function Class:

Not an API, this function is for internal use only.

9.7.2.37 #define currp ch.list.r_current

Current thread pointer access macro.

Note

This macro is not meant to be used in the application code but only from within the kernel, use the `chThdSelf()` API instead.

It is forbidden to use this macro in order to change the pointer (`currp = something`), use `setcurrp()` instead.

9.7.2.38 #define setcurrp(tp) (currp = (tp))

Current thread pointer change macro.

Note

This macro is not meant to be used in the application code but only from within the kernel.

Function Class:

Not an API, this function is for internal use only.

9.7.3 Typedef Documentation**9.7.3.1 typedef uint32_t systime_t**

Type of system time.

9.7.3.2 typedef struct ch_thread thread_t

Type of a thread structure.

9.7.3.3 typedef thread_t* thread_reference_t

Type of a thread reference.

9.7.3.4 typedef struct ch_threads_list threads_list_t

Type of a generic threads single link list, it works like a stack.

9.7.3.5 typedef struct ch_threads_queue threads_queue_t

Type of a generic threads bidirectional linked list header and element.

9.7.3.6 typedef struct ch_ready_list ready_list_t

Type of a ready list header.

9.7.3.7 typedef void(* vfunc_t)(void *p)

Type of a Virtual Timer callback function.

9.7.3.8 `typedef struct ch_virtual_timer virtual_timer_t`

Type of a Virtual Timer structure.

9.7.3.9 `typedef struct ch_virtual_timers_list virtual_timers_list_t`

Type of virtual timers list header.

9.7.3.10 `typedef struct ch_system_debug system_debug_t`

Type of a system debug structure.

9.7.3.11 `typedef struct ch_system ch_system_t`

Type of system data structure.

9.7.4 Function Documentation

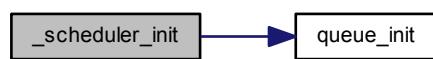
9.7.4.1 `void _scheduler_init(void)`

Scheduler initialization.

Function Class:

Not an API, this function is for internal use only.

Here is the call graph for this function:



9.7.4.2 `static void queue_prio_insert(thread_t * tp, threads_queue_t * tqp) [inline]`

Inserts a thread into a priority ordered queue.

Note

The insertion is done by scanning the list from the highest priority toward the lowest.

Parameters

in	<i>tp</i>	the pointer to the thread to be inserted in the list
----	-----------	--

in	<i>tqp</i>	the pointer to the threads list header
----	------------	--

Function Class:

Not an API, this function is for internal use only.

9.7.4.3 static void queue_insert (*thread_t* * *tp*, *threads_queue_t* * *tqp*) [inline]

Inserts a thread into a queue.

Parameters

in	<i>tp</i>	the pointer to the thread to be inserted in the list
in	<i>tqp</i>	the pointer to the threads list header

Function Class:

Not an API, this function is for internal use only.

9.7.4.4 static *thread_t* * queue_fifo_remove (*threads_queue_t* * *tqp*) [inline]

Removes the first-out thread from a queue and returns it.

Note

If the queue is priority ordered then this function returns the thread with the highest priority.

Parameters

in	<i>tqp</i>	the pointer to the threads list header
----	------------	--

Returns

The removed thread pointer.

Function Class:

Not an API, this function is for internal use only.

9.7.4.5 static *thread_t* * queue_lifo_remove (*threads_queue_t* * *tqp*) [inline]

Removes the last-out thread from a queue and returns it.

Note

If the queue is priority ordered then this function returns the thread with the lowest priority.

Parameters

in	<i>tqp</i>	the pointer to the threads list header
----	------------	--

Returns

The removed thread pointer.

Function Class:

Not an API, this function is for internal use only.

9.7.4.6 static `thread_t * queue_dequeue(thread_t * tp) [inline]`

Removes a thread from a queue and returns it.

The thread is removed from the queue regardless of its relative position and regardless the used insertion method.

Parameters

in	<i>tp</i>	the pointer to the thread to be removed from the queue
----	-----------	--

Returns

The removed thread pointer.

Function Class:

Not an API, this function is for internal use only.

9.7.4.7 static void `list_insert(thread_t * tp, threads_list_t * tlp) [inline]`

Pushes a `thread_t` on top of a stack list.

Parameters

in	<i>tp</i>	the pointer to the thread to be inserted in the list
in	<i>tlp</i>	the pointer to the threads list header

Function Class:

Not an API, this function is for internal use only.

9.7.4.8 static `thread_t * list_remove(threads_list_t * tlp) [inline]`

Pops a thread from the top of a stack list and returns it.

Precondition

The list must be non-empty before calling this function.

Parameters

in	<i>tlp</i>	the pointer to the threads list header
----	------------	--

Returns

The removed thread pointer.

Function Class:

Not an API, this function is for internal use only.

9.7.4.9 `thread_t * chSchReady(thread_t * tp)`

Inserts a thread in the Ready List.

The thread is positioned behind all threads with higher or equal priority.

Precondition

The thread must not be already inserted in any list through its `p_next` and `p_prev` or list corruption would occur.

Postcondition

This function does not reschedule so a call to a rescheduling function must be performed before unlocking the kernel. Note that interrupt handlers always reschedule on exit so an explicit reschedule must not be performed in ISRs.

Parameters

in	<i>tp</i>	the thread to be made ready
----	-----------	-----------------------------

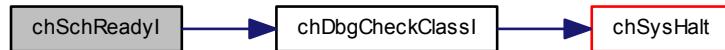
Returns

The thread pointer.

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:

**9.7.4.10 void chSchGoSleepS (tstate_t newstate)**

Puts the current thread to sleep into the specified state.

The thread goes into a sleeping state. The possible **Thread States** are defined into `threads.h`.

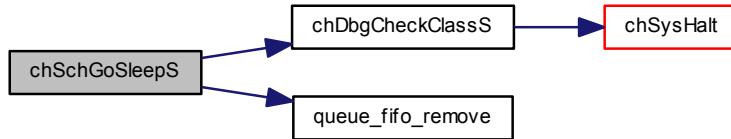
Parameters

in	<i>newstate</i>	the new thread state
----	-----------------	----------------------

Function Class:

This is an **S-Class** API, this function can be invoked from within a system lock zone by threads only.

Here is the call graph for this function:



9.7.4.11 msg_t chSchGoSleepTimeoutS (tstate_t newstate, systime_t time)

Puts the current thread to sleep into the specified state with timeout specification.

The thread goes into a sleeping state, if it is not awakened explicitly within the specified timeout then it is forcibly awakened with a `MSG_TIMEOUT` low level message. The possible [Thread States](#) are defined into `threads.h`.

Parameters

in	<i>newstate</i>	the new thread state
in	<i>time</i>	<p>the number of ticks before the operation timeouts, the special values are handled as follow:</p> <ul style="list-style-type: none"> • <code>TIME_INFINITE</code> the thread enters an infinite sleep state, this is equivalent to invoking <code>chSchGoSleepS()</code> but, of course, less efficient. • <code>TIME_IMMEDIATE</code> this value is not allowed.

Returns

The wakeup message.

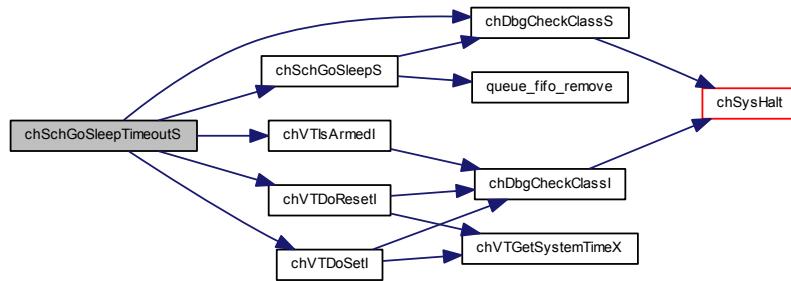
Return values

<code>MSG_TIMEOUT</code>	if a timeout occurs.
--------------------------	----------------------

Function Class:

This is an **S-Class** API, this function can be invoked from within a system lock zone by threads only.

Here is the call graph for this function:



9.7.4.12 void chSchWakeupS (thread_t *ntp, msg_t msg)

Wakes up a thread.

The thread is inserted into the ready list or immediately made running depending on its relative priority compared to the current thread.

Precondition

The thread must not be already inserted in any list through its `p_next` and `p_prev` or list corruption would occur.

Note

It is equivalent to a `chSchReadyI()` followed by a `chSchRescheduleS()` but much more efficient.
The function assumes that the current thread has the highest priority.

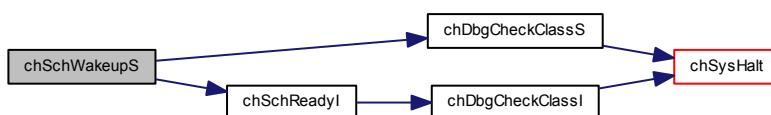
Parameters

in	<code>ntp</code>	the thread to be made ready
in	<code>msg</code>	the wakeup message

Function Class:

This is an **S-Class** API, this function can be invoked from within a system lock zone by threads only.

Here is the call graph for this function:



9.7.4.13 void chSchRescheduleS (void)

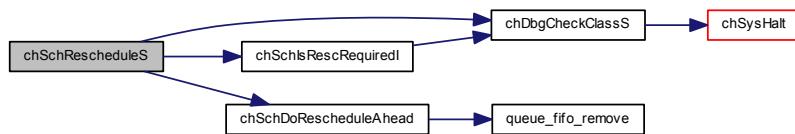
Performs a reschedule if a higher priority thread is runnable.

If a thread with a higher priority than the current thread is in the ready list then make the higher priority thread running.

Function Class:

This is an **S-Class** API, this function can be invoked from within a system lock zone by threads only.

Here is the call graph for this function:



9.7.4.14 bool chSchIsPreemptionRequired (void)

Evaluates if preemption is required.

The decision is taken by comparing the relative priorities and depending on the state of the round robin timeout counter.

Note

Not a user function, it is meant to be invoked by the scheduler itself or from within the port layer.

Return values

<i>true</i>	if there is a thread that must go in running state immediately.
<i>false</i>	if preemption is not required.

Function Class:

Special function, this function has special requirements see the notes.

9.7.4.15 void chSchDoRescheduleBehind (void)

Switches to the first thread on the runnable queue.

The current thread is positioned in the ready list behind all threads having the same priority. The thread regains its time quantum.

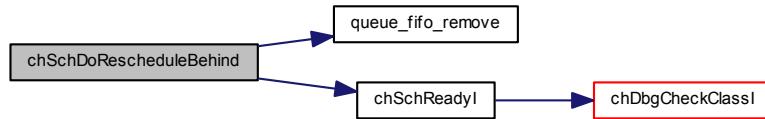
Note

Not a user function, it is meant to be invoked by the scheduler itself or from within the port layer.

Function Class:

Special function, this function has special requirements see the notes.

Here is the call graph for this function:



9.7.4.16 void chSchDoRescheduleAhead (void)

Switches to the first thread on the runnable queue.

The current thread is positioned in the ready list ahead of all threads having the same priority.

Note

Not a user function, it is meant to be invoked by the scheduler itself or from within the port layer.

Function Class:

Special function, this function has special requirements see the notes.

Here is the call graph for this function:



9.7.4.17 void chSchDoReschedule (void)

Switches to the first thread on the runnable queue.

The current thread is positioned in the ready list behind or ahead of all threads having the same priority depending on if it used its whole time slice.

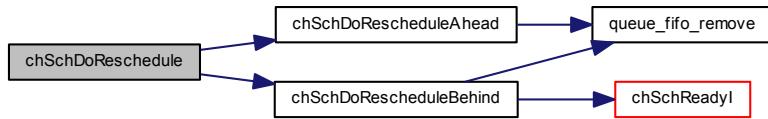
Note

Not a user function, it is meant to be invoked by the scheduler itself or from within the port layer.

Function Class:

Special function, this function has special requirements see the notes.

Here is the call graph for this function:



9.7.4.18 static void list_init(threads_list_t * tlp) [inline], [static]

Threads list initialization.

Parameters

in	<i>tlp</i>	pointer to the threads list object
----	------------	------------------------------------

Function Class:

Not an API, this function is for internal use only.

9.7.4.19 static bool list_isempty(threads_list_t * tlp) [inline], [static]

Evaluates to `true` if the specified threads list is empty.

Parameters

in	<i>tlp</i>	pointer to the threads list object
----	------------	------------------------------------

Returns

The status of the list.

Function Class:

Not an API, this function is for internal use only.

9.7.4.20 static bool list_notempty(threads_list_t * tlp) [inline], [static]

Evaluates to `true` if the specified threads list is not empty.

Parameters

in	<i>tlp</i>	pointer to the threads list object
----	------------	------------------------------------

Returns

The status of the list.

Function Class:

Not an API, this function is for internal use only.

9.7.4.21 static void queue_init(threads_queue_t * *tqp*) [inline], [static]

Threads queue initialization.

Parameters

in	<i>tqp</i>	pointer to the threads queue object
----	------------	-------------------------------------

Function Class:

Not an API, this function is for internal use only.

9.7.4.22 static bool queue_isempty (const threads_queue_t * *tqp*) [inline], [static]

Evaluates to `true` if the specified threads queue is empty.

Parameters

in	<i>tqp</i>	pointer to the threads queue object
----	------------	-------------------------------------

Returns

The status of the queue.

Function Class:

Not an API, this function is for internal use only.

9.7.4.23 static bool queue_notempty (const threads_queue_t * *tqp*) [inline], [static]

Evaluates to `true` if the specified threads queue is not empty.

Parameters

in	<i>tqp</i>	pointer to the threads queue object
----	------------	-------------------------------------

Returns

The status of the queue.

Function Class:

Not an API, this function is for internal use only.

9.7.4.24 static bool chSchIsRescRequired(void) [inline], [static]

Determines if the current thread must reschedule.

This function returns `true` if there is a ready thread with higher priority.

Returns

The priorities situation.

Return values

<i>false</i>	if rescheduling is not necessary.
--------------	-----------------------------------

<code>true</code>	if there is a ready thread at higher priority.
-------------------	--

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:

**9.7.4.25 static bool chSchCanYieldS(void) [inline], [static]**

Determines if yielding is possible.

This function returns `true` if there is a ready thread with equal or higher priority.

Returns

The priorities situation.

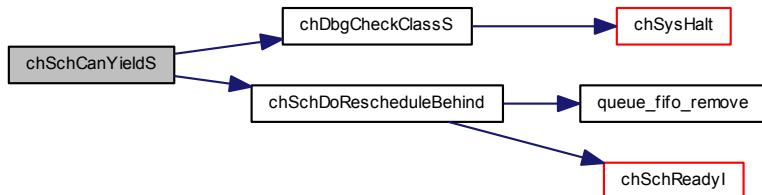
Return values

<code>false</code>	if yielding is not possible.
<code>true</code>	if there is a ready thread at equal or higher priority.

Function Class:

This is an **S-Class** API, this function can be invoked from within a system lock zone by threads only.

Here is the call graph for this function:

**9.7.4.26 static void chSchDoYieldS(void) [inline], [static]**

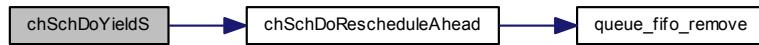
Yields the time slot.

Yields the CPU control to the next thread in the ready list with equal or higher priority, if any.

Function Class:

This is an **S-Class** API, this function can be invoked from within a system lock zone by threads only.

Here is the call graph for this function:



9.7.4.27 static void chSchPreemption(void) [inline], [static]

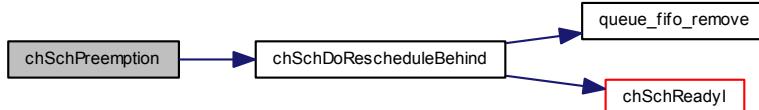
Inline-able preemption code.

This is the common preemption code, this function must be invoked exclusively from the port layer.

Function Class:

Special function, this function has special requirements see the notes.

Here is the call graph for this function:



9.7.5 Variable Documentation

9.7.5.1 ch_system_t ch

System data structures.

9.8 Threads

9.8.1 Detailed Description

Threads related APIs and services.

Operation mode

A thread is an abstraction of an independent instructions flow. In ChibiOS/RT a thread is represented by a "C" function owning a processor context, state informations and a dedicated stack area. In this scenario static variables are shared among all threads while automatic variables are local to the thread.

Operations defined for threads:

- **Create**, a thread is started on the specified thread function. This operation is available in multiple variants, both static and dynamic.
- **Exit**, a thread terminates by returning from its top level function or invoking a specific API, the thread can return a value that can be retrieved by other threads.
- **Wait**, a thread waits for the termination of another thread and retrieves its return value.
- **Resume**, a thread created in suspended state is started.
- **Sleep**, the execution of a thread is suspended for the specified amount of time or the specified future absolute time is reached.
- **SetPriority**, a thread changes its own priority level.
- **Yield**, a thread voluntarily renounces to its time slot.

The threads subsystem is implicitly included in kernel however some of its part may be excluded by disabling them in `chconf.h`, see the `CH_CFG_USE_WAITEXIT` and `CH_CFG_USE_DYNAMIC` configuration options.

Threads queues

- `#define _THREADS_QUEUE_DATA(name) {(<thread_t *>)name, (<thread_t *>)name}`
Data part of a static threads queue object initializer.
- `#define _THREADS_QUEUE_DECL(name) threads_queue_t name = _THREADS_QUEUE_DATA(name)`
Static threads queue object initializer.

Macro Functions

- `#define chThdSleepSeconds(sec) chThdSleep(S2ST(sec))`
Delays the invoking thread for the specified number of seconds.
- `#define chThdSleepMilliseconds(msec) chThdSleep(MS2ST(msec))`
Delays the invoking thread for the specified number of milliseconds.
- `#define chThdSleepMicroseconds(usec) chThdSleep(US2ST(usec))`
Delays the invoking thread for the specified number of microseconds.

Typedefs

- `typedef void(* tfunc_t) (void *p)`
Thread function.

Functions

- **`thread_t * _thread_init (thread_t *tp, tprio_t prio)`**
Initializes a thread structure.
- **`void _thread_memfill (uint8_t *startp, uint8_t *endp, uint8_t v)`**
Memory fill utility.
- **`thread_t * chThdCreateI (void *wsp, size_t size, tprio_t prio, tfunc_t pf, void *arg)`**
Creates a new thread into a static memory area.
- **`thread_t * chThdCreateStatic (void *wsp, size_t size, tprio_t prio, tfunc_t pf, void *arg)`**
Creates a new thread into a static memory area.
- **`thread_t * chThdStart (thread_t *tp)`**
Resumes a thread created with `chThdCreateI ()`.
- **`tprio_t chThdSetPriority (tprio_t newprio)`**
Changes the running thread priority level then reschedules if necessary.
- **`void chThdTerminate (thread_t *tp)`**
Requests a thread termination.
- **`void chThdSleep (systime_t time)`**
Suspends the invoking thread for the specified time.
- **`void chThdSleepUntil (systime_t time)`**
Suspends the invoking thread until the system time arrives to the specified value.
- **`systime_t chThdSleepUntilWindowed (systime_t prev, systime_t next)`**
Suspends the invoking thread until the system time arrives to the specified value.
- **`void chThdYield (void)`**
Yields the time slot.
- **`void chThdExit (msg_t msg)`**
Terminates the current thread.
- **`void chThdExitS (msg_t msg)`**
Terminates the current thread.
- **`msg_t chThdWait (thread_t *tp)`**
Blocks the execution of the invoking thread until the specified thread terminates then the exit code is returned.
- **`msg_t chThdSuspendS (thread_reference_t *trp)`**
Sends the current thread sleeping and sets a reference variable.
- **`msg_t chThdSuspendTimeoutS (thread_reference_t *trp, systime_t timeout)`**
Sends the current thread sleeping and sets a reference variable.
- **`void chThdResumel (thread_reference_t *trp, msg_t msg)`**
Wakes up a thread waiting on a thread reference object.
- **`void chThdResumeS (thread_reference_t *trp, msg_t msg)`**
Wakes up a thread waiting on a thread reference object.
- **`void chThdResume (thread_reference_t *trp, msg_t msg)`**
Wakes up a thread waiting on a thread reference object.
- **`msg_t chThdEnqueueTimeoutS (threads_queue_t *tqp, systime_t timeout)`**
Enqueues the caller thread on a threads queue object.
- **`void chThdDequeueNextI (threads_queue_t *tqp, msg_t msg)`**
Dequeues and wakes up one thread from the threads queue object, if any.
- **`void chThdDequeueAllI (threads_queue_t *tqp, msg_t msg)`**
Dequeues and wakes up all threads from the threads queue object.
- **`static thread_t * chThdGetSelfX (void)`**
Returns a pointer to the current `thread_t`.
- **`static tprio_t chThdGetPriorityX (void)`**
Returns the current thread priority.
- **`static systime_t chThdGetTicksX (thread_t *tp)`**

- static bool `chThdTerminatedX` (`thread_t` *`tp`)

Returns the number of ticks consumed by the specified thread.
- static bool `chThdShouldTerminateX` (`void`)

Verifies if the specified thread is in the CH_STATE_FINAL state.
- static `thread_t` * `chThdStartI` (`thread_t` *`tp`)

Resumes a thread created with `chThdCreateI()`.
- static void `chThdSleepS` (`systime_t` `time`)

Suspends the invoking thread for the specified time.
- static void `chThdQueueObjectInit` (`threads_queue_t` *`tqp`)

Initializes a threads queue object.
- static bool `chThdQueueIsEmpty` (`threads_queue_t` *`tqp`)

Evaluates to true if the specified queue is empty.
- static void `chThdDoDequeueNextI` (`threads_queue_t` *`tqp`, `msg_t` `msg`)

Dequeues and wakes up one thread from the threads queue object.

9.8.2 Macro Definition Documentation

9.8.2.1 #define _THREADS_QUEUE_DATA(`name`) {(`thread_t` *)`&name`, (`thread_t` *)`&name`}

Data part of a static threads queue object initializer.

This macro should be used when statically initializing a threads queue that is part of a bigger structure.

Parameters

in	<code>name</code>	the name of the threads queue variable
----	-------------------	--

9.8.2.2 #define _THREADS_QUEUE_DECL(`name`) `threads_queue_t` `name` = _THREADS_QUEUE_DATA(`name`)

Static threads queue object initializer.

Statically initialized threads queues require no explicit initialization using `queue_init()`.

Parameters

in	<code>name</code>	the name of the threads queue variable
----	-------------------	--

9.8.2.3 #define chThdSleepSeconds(`sec`) chThdSleep(S2ST(`sec`))

Delays the invoking thread for the specified number of seconds.

Note

The specified time is rounded up to a value allowed by the real system tick clock.
The maximum specifiable value is implementation dependent.

Parameters

in	<code>sec</code>	time in seconds, must be different from zero
----	------------------	--

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

9.8.2.4 #define chThdSleepMilliseconds(*msec*) chThdSleep(MS2ST(*msec*))

Delays the invoking thread for the specified number of milliseconds.

Note

The specified time is rounded up to a value allowed by the real system tick clock.
The maximum specifiable value is implementation dependent.

Parameters

in	<i>msec</i>	time in milliseconds, must be different from zero
----	-------------	---

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

9.8.2.5 #define chThdSleepMicroseconds(*usec*) chThdSleep(US2ST(*usec*))

Delays the invoking thread for the specified number of microseconds.

Note

The specified time is rounded up to a value allowed by the real system tick clock.
The maximum specifiable value is implementation dependent.

Parameters

in	<i>usec</i>	time in microseconds, must be different from zero
----	-------------	---

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

9.8.3 Typedef Documentation

9.8.3.1 `typedef void(* tfunc_t)(void *p)`

Thread function.

9.8.4 Function Documentation

9.8.4.1 `thread_t *_thread_init(thread_t *tp, tprio_t prio)`

Initializes a thread structure.

Note

This is an internal functions, do not use it in application code.

Parameters

in	<i>tp</i>	pointer to the thread
in	<i>prio</i>	the priority level for the new thread

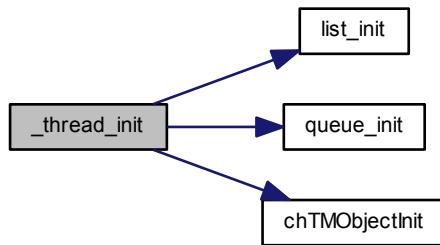
Returns

The same thread pointer passed as parameter.

Function Class:

Not an API, this function is for internal use only.

Here is the call graph for this function:

**9.8.4.2 void _thread_memfill(uint8_t * startp, uint8_t * endp, uint8_t v)**

Memory fill utility.

Parameters

in	<i>startp</i>	first address to fill
in	<i>endp</i>	last address to fill +1
in	<i>v</i>	filler value

Function Class:

Not an API, this function is for internal use only.

9.8.4.3 thread_t * chThdCreateL(void * wsp, size_t size, tprio_t prio, tfunc_t pf, void * arg)

Creates a new thread into a static memory area.

The new thread is initialized but not inserted in the ready list, the initial state is CH_STATE_WTSTART.

Postcondition

The initialized thread can be subsequently started by invoking [chThdStart\(\)](#), [chThdStartI\(\)](#) or [chSchWakeupS\(\)](#) depending on the execution context.

Note

A thread can terminate by calling [chThdExit\(\)](#) or by simply returning from its main function.

Threads created using this function do not obey to the CH_DBG_FILL_THREADS debug option because it would keep the kernel locked for too much time.

Parameters

out	<i>wsp</i>	pointer to a working area dedicated to the thread stack
in	<i>size</i>	size of the working area
in	<i>prio</i>	the priority level for the new thread
in	<i>pf</i>	the thread function
in	<i>arg</i>	an argument passed to the thread function. It can be NULL.

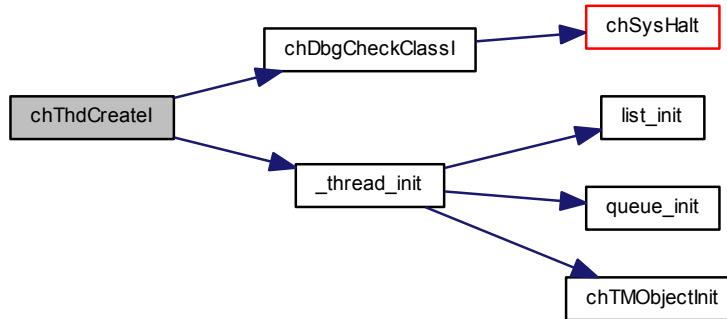
Returns

The pointer to the `thread_t` structure allocated for the thread into the working space area.

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:



9.8.4.4 `thread_t * chThdCreateStatic(void * wsp, size_t size, tprio_t prio, tfunc_t pf, void * arg)`

Creates a new thread into a static memory area.

Note

A thread can terminate by calling `chThdExit()` or by simply returning from its main function.

Parameters

out	<i>wsp</i>	pointer to a working area dedicated to the thread stack
in	<i>size</i>	size of the working area
in	<i>prio</i>	the priority level for the new thread
in	<i>pf</i>	the thread function
in	<i>arg</i>	an argument passed to the thread function. It can be NULL.

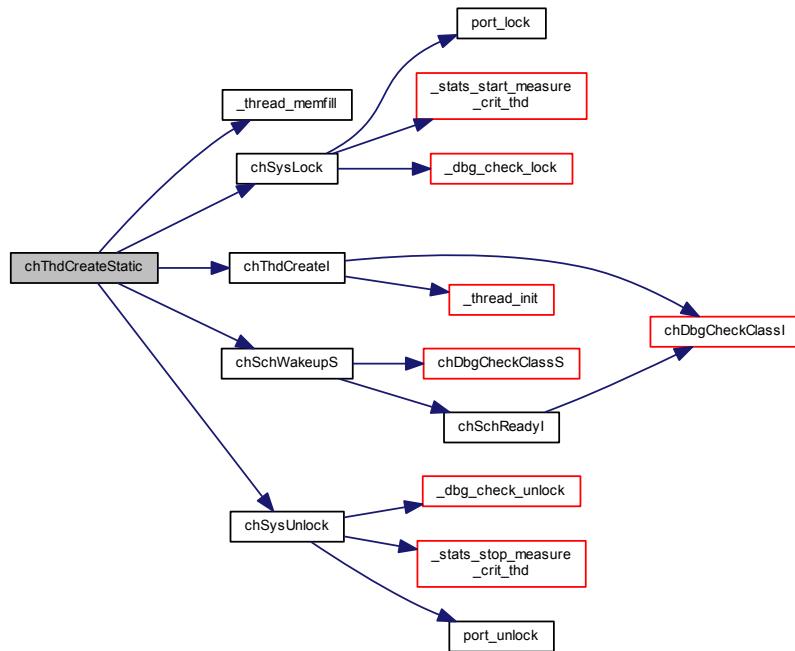
Returns

The pointer to the `thread_t` structure allocated for the thread into the working space area.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

**9.8.4.5 `thread_t * chThdStart(thread_t * tp)`**

Resumes a thread created with `chThdCreateI()`.

Parameters

in	<code>tp</code>	pointer to the thread
----	-----------------	-----------------------

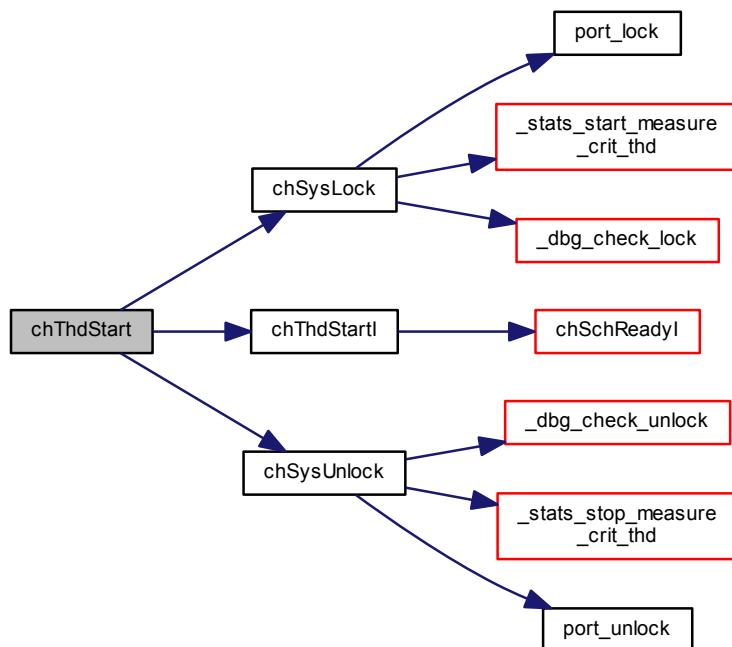
Returns

The pointer to the `thread_t` structure allocated for the thread into the working space area.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

**9.8.4.6 `tprio_t chThdSetPriority (tprio_t newprio)`**

Changes the running thread priority level then reschedules if necessary.

Note

The function returns the real thread priority regardless of the current priority that could be higher than the real priority because the priority inheritance mechanism.

Parameters

in	<code>newprio</code>	the new priority level of the running thread
----	----------------------	--

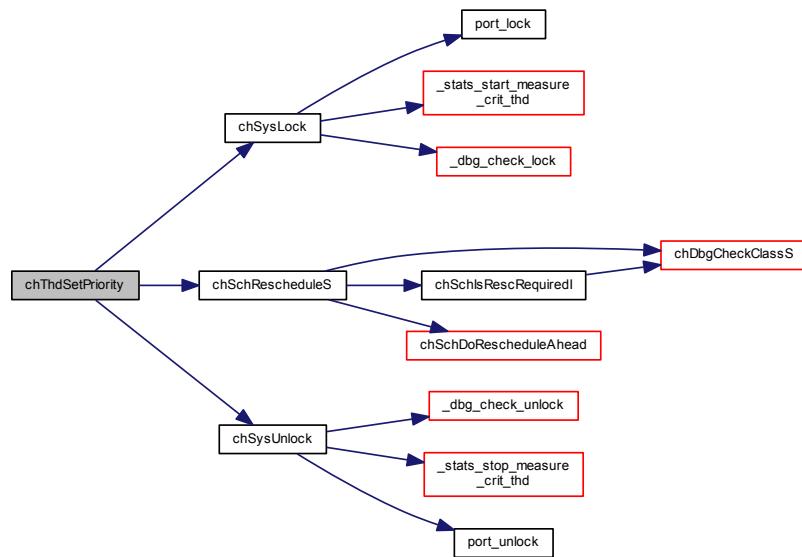
Returns

The old priority level.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



9.8.4.7 void chThdTerminate (`thread_t * tp`)

Requests a thread termination.

Precondition

The target thread must be written to invoke periodically `chThdShouldTerminate()` and terminate cleanly if it returns `true`.

Postcondition

The specified thread will terminate after detecting the termination condition.

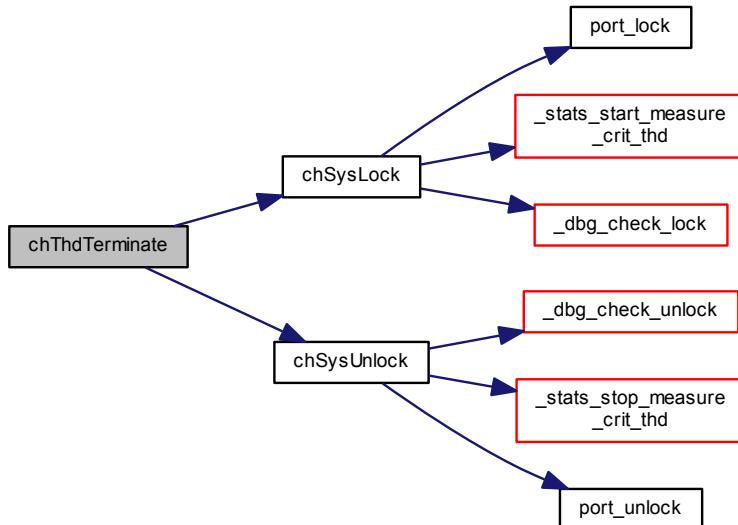
Parameters

in	<i>tp</i>	pointer to the thread
----	-----------	-----------------------

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

**9.8.4.8 void chThdSleep(systime_t time)**

Suspends the invoking thread for the specified time.

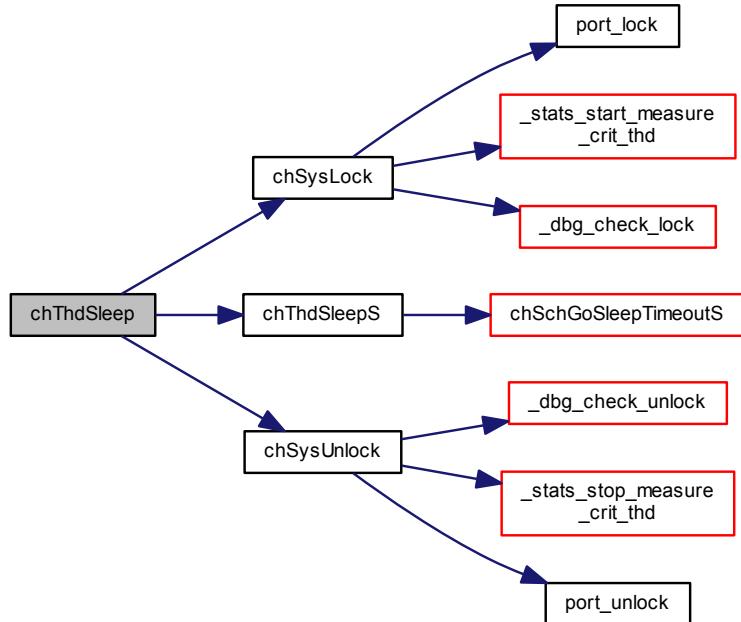
Parameters

in	<i>time</i>	the delay in system ticks, the special values are handled as follow: <ul style="list-style-type: none"> <i>TIME_INFINITE</i> the thread enters an infinite sleep state. <i>TIME_IMMEDIATE</i> this value is not allowed.
----	-------------	--

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



9.8.4.9 void chThdSleepUntil (systime_t time)

Suspends the invoking thread until the system time arrives to the specified value.

Note

The function has no concept of "past", all specifiable times are in the future, this means that if you call this function exceeding your calculated intervals then the function will return in a far future time, not immediately.

See also

[chThdSleepUntilWindowed\(\)](#)

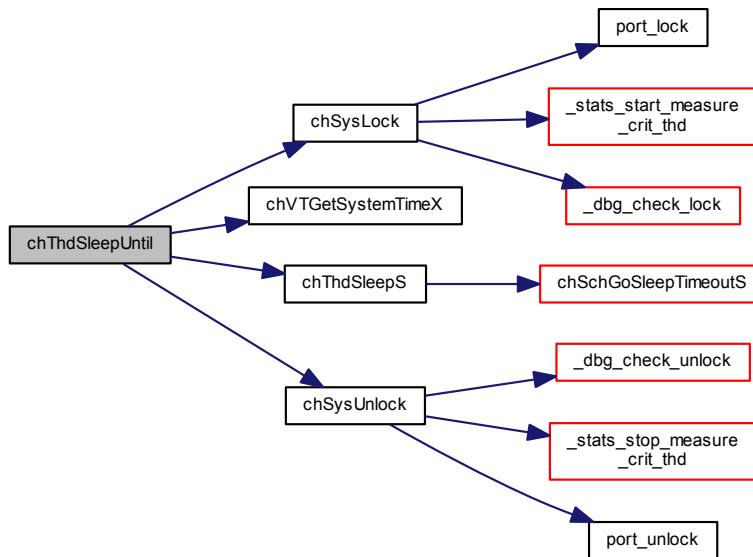
Parameters

in	time	absolute system time
----	------	----------------------

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



9.8.4.10 `systime_t chThdSleepUntilWindowed(systime_t prev, systime_t next)`

Suspends the invoking thread until the system time arrives to the specified value.

Note

The system time is assumed to be between `prev` and `time` else the call is assumed to have been called outside the allowed time interval, in this case no sleep is performed.

See also

[chThdSleepUntil\(\)](#)

Parameters

in	prev	absolute system time of the previous deadline
in	next	absolute system time of the next deadline

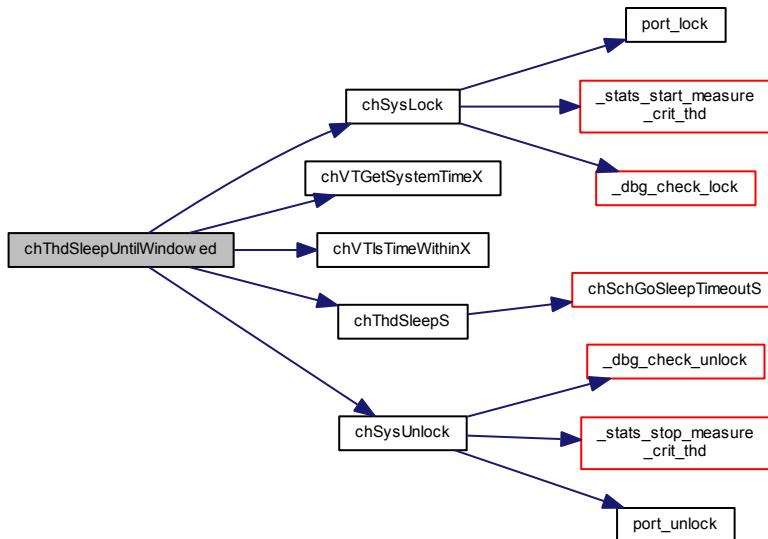
Returns

the next parameter

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

**9.8.4.11 void chThdYield (void)**

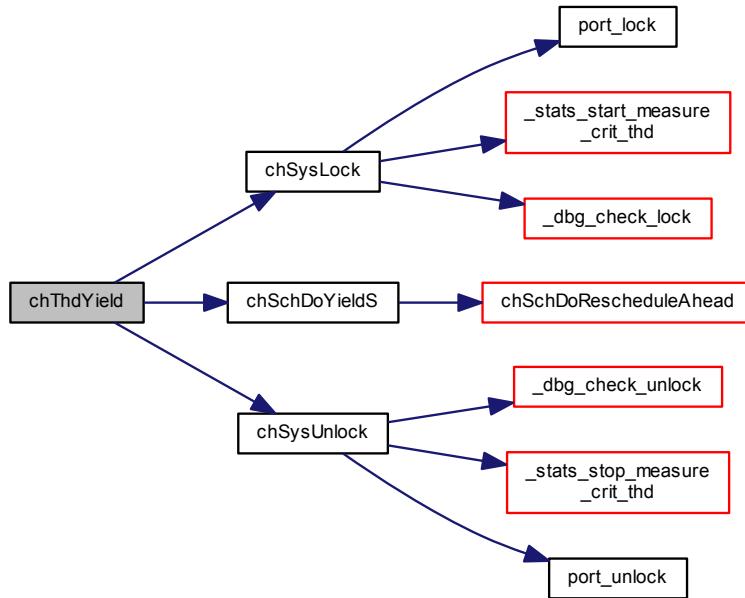
Yields the time slot.

Yields the CPU control to the next thread in the ready list with equal priority, if any.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



9.8.4.12 void chThdExit (msg_t msg)

Terminates the current thread.

The thread goes in the `CH_STATE_FINAL` state holding the specified exit status code, other threads can retrieve the exit status code by invoking the function `chThdWait()`.

Postcondition

Eventual code after this function will never be executed, this function never returns. The compiler has no way to know this so do not assume that the compiler would remove the dead code.

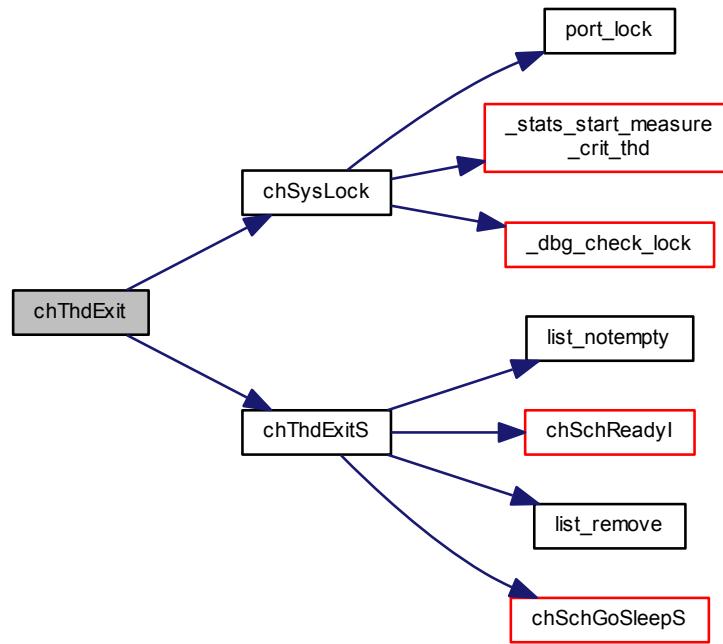
Parameters

in	<code>msg</code>	thread exit code
----	------------------	------------------

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



9.8.4.13 void chThdExitS (msg_t msg)

Terminates the current thread.

The thread goes in the CH_STATE_FINAL state holding the specified exit status code, other threads can retrieve the exit status code by invoking the function `chThdWait()`.

Postcondition

Eventual code after this function will never be executed, this function never returns. The compiler has no way to know this so do not assume that the compiler would remove the dead code.

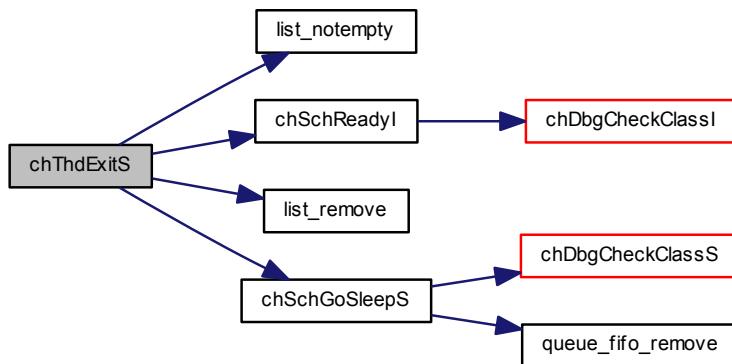
Parameters

in	msg	thread exit code
----	-----	------------------

Function Class:

This is an **S-Class** API, this function can be invoked from within a system lock zone by threads only.

Here is the call graph for this function:



9.8.4.14 msg_t chThdWait(thread_t * tp)

Blocks the execution of the invoking thread until the specified thread terminates then the exit code is returned.

This function waits for the specified thread to terminate then decrements its reference counter, if the counter reaches zero then the thread working area is returned to the proper allocator.

The memory used by the exited thread is handled in different ways depending on the API that spawned the thread:

- If the thread was spawned by `chThdCreateStatic()` or by `chThdCreateI()` then nothing happens and the thread working area is not released or modified in any way. This is the default, totally static, behavior.
- If the thread was spawned by `chThdCreateFromHeap()` then the working area is returned to the system heap.
- If the thread was spawned by `chThdCreateFromMemoryPool()` then the working area is returned to the owning memory pool.

Precondition

The configuration option `CH_CFG_USE_WAITEXIT` must be enabled in order to use this function.

Postcondition

Enabling `chThdWait()` requires 2-4 (depending on the architecture) extra bytes in the `thread_t` structure.

After invoking `chThdWait()` the thread pointer becomes invalid and must not be used as parameter for further system calls.

Note

If `CH_CFG_USE_DYNAMIC` is not specified this function just waits for the thread termination, no memory allocators are involved.

Parameters

in	<i>tp</i>	pointer to the thread
----	-----------	-----------------------

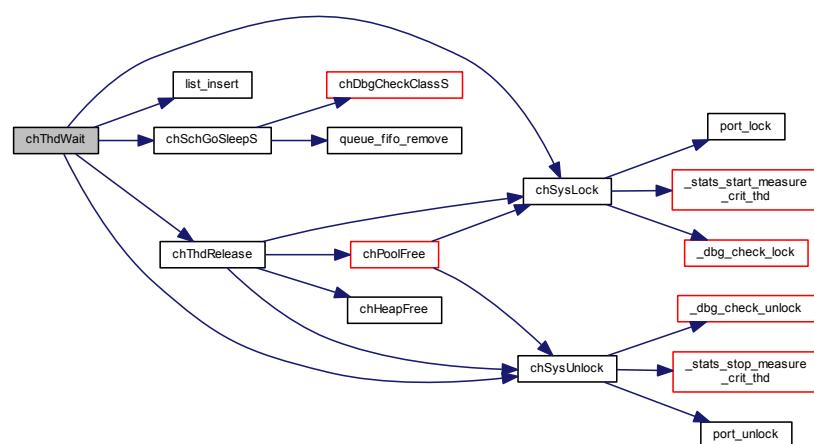
Returns

The exit code from the terminated thread.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

**9.8.4.15 msg_t chThdSuspendS (thread_reference_t * trp)**

Sends the current thread sleeping and sets a reference variable.

Note

This function must reschedule, it can only be called from thread context.

Parameters

in	<i>trp</i>	a pointer to a thread reference object
----	------------	--

Returns

The wake up message.

Function Class:

This is an **S-Class** API, this function can be invoked from within a system lock zone by threads only.

Here is the call graph for this function:

**9.8.4.16 msg_t chThdSuspendTimeoutS(thread_reference_t * trp, systime_t timeout)**

Sends the current thread sleeping and sets a reference variable.

Note

This function must reschedule, it can only be called from thread context.

Parameters

in	<i>trp</i>	a pointer to a thread reference object
in	<i>timeout</i>	the timeout in system ticks, the special values are handled as follow: <ul style="list-style-type: none"> • <i>TIME_INFINITE</i> the thread enters an infinite sleep state. • <i>TIME_IMMEDIATE</i> the thread is not enqueued and the function returns <i>MSG_TIMEOUT</i> as if a timeout occurred.

Returns

The wake up message.

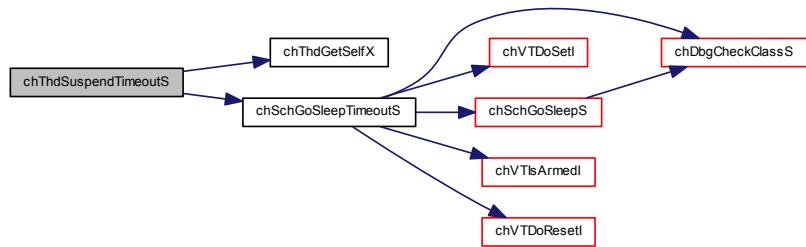
Return values

<i>MSG_TIMEOUT</i>	if the operation timed out.
--------------------	-----------------------------

Function Class:

This is an **S-Class** API, this function can be invoked from within a system lock zone by threads only.

Here is the call graph for this function:



9.8.4.17 void chThdResumel (*thread_reference_t* * *trp*, *msg_t* *msg*)

Wakes up a thread waiting on a thread reference object.

Note

This function must not reschedule because it can be called from ISR context.

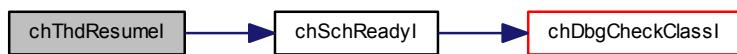
Parameters

in	<i>trp</i>	a pointer to a thread reference object
in	<i>msg</i>	the message code

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:



9.8.4.18 void chThdResumeS (*thread_reference_t* * *trp*, *msg_t* *msg*)

Wakes up a thread waiting on a thread reference object.

Note

This function must reschedule, it can only be called from thread context.

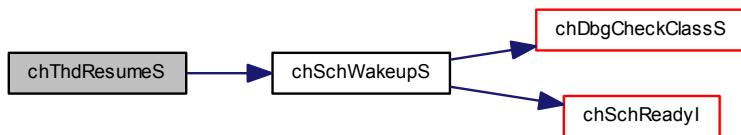
Parameters

in	<i>trp</i>	a pointer to a thread reference object
in	<i>msg</i>	the message code

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:



9.8.4.19 void chThdResume(thread_reference_t * *trp*, msg_t *msg*)

Wakes up a thread waiting on a thread reference object.

Note

This function must reschedule, it can only be called from thread context.

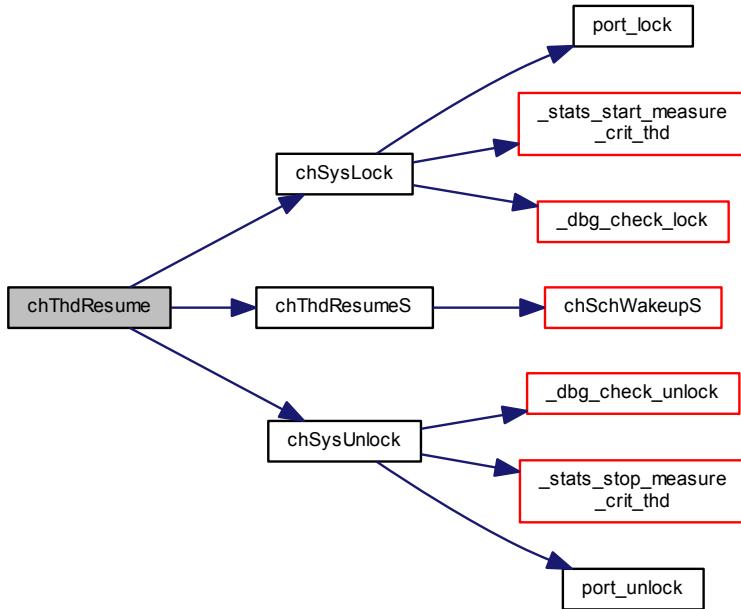
Parameters

in	<i>trp</i>	a pointer to a thread reference object
in	<i>msg</i>	the message code

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



9.8.4.20 msg_t chThdEnqueueTimeoutS (threads_queue_t * tqp, systime_t timeout)

Enqueues the caller thread on a threads queue object.

The caller thread is enqueued and put to sleep until it is dequeued or the specified timeouts expires.

Parameters

in	<i>tqp</i>	pointer to the threads queue object
in	<i>timeout</i>	the timeout in system ticks, the special values are handled as follow: <ul style="list-style-type: none"> • <i>TIME_INFINITE</i> the thread enters an infinite sleep state. • <i>TIME_IMMEDIATE</i> the thread is not enqueued and the function returns <code>MSG_TIMEOUT</code> as if a timeout occurred.

Returns

The message from `osalQueueWakeupOneI()` or `osalQueueWakeupAllI()` functions.

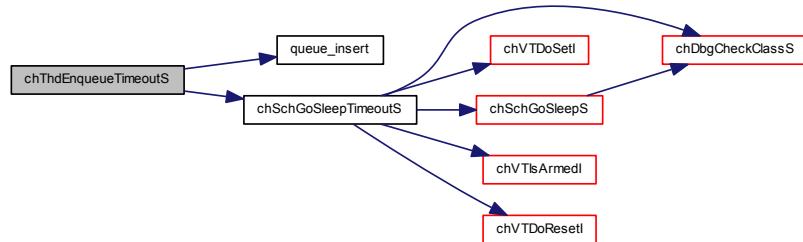
Return values

<code>MSG_TIMEOUT</code>	if the thread has not been dequeued within the specified timeout or if the function has been invoked with <code>TIME_IMMEDIATE</code> as timeout specification.
--------------------------	---

Function Class:

This is an **S-Class** API, this function can be invoked from within a system lock zone by threads only.

Here is the call graph for this function:



9.8.4.21 void chThdDequeueNextI (threads_queue_t * tqp, msg_t msg)

Dequeues and wakes up one thread from the threads queue object, if any.

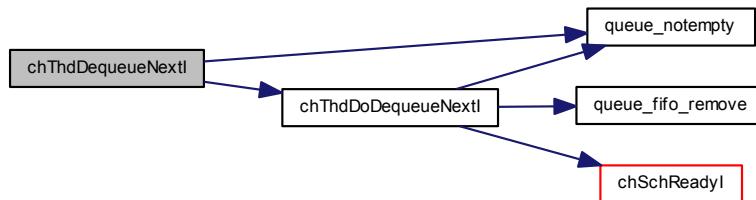
Parameters

in	<i>tqp</i>	pointer to the threads queue object
in	<i>msg</i>	the message code

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:



9.8.4.22 void chThdDequeueAllI (threads_queue_t * tqp, msg_t msg)

Dequeues and wakes up all threads from the threads queue object.

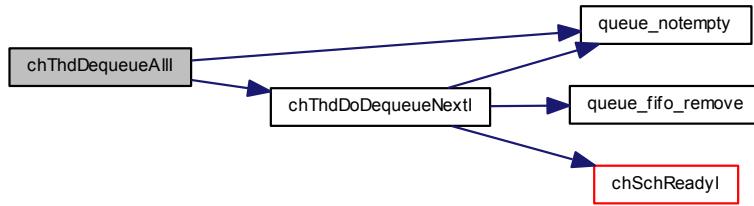
Parameters

in	<i>tqp</i>	pointer to the threads queue object
in	<i>msg</i>	the message code

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:



9.8.4.23 static thread_t* chThdGetSelfX(void) [inline], [static]

Returns a pointer to the current `thread_t`.

Returns

A pointer to the current thread.

Function Class:

This is an **X-Class** API, this function can be invoked from any context.

9.8.4.24 static tprio_t chThdGetPriorityX(void) [inline], [static]

Returns the current thread priority.

Note

Can be invoked in any context.

Returns

The current thread priority.

Function Class:

This is an **X-Class** API, this function can be invoked from any context.

Here is the call graph for this function:



9.8.4.25 static systime_t chThdGetTicksX(thread_t * tp) [inline], [static]

Returns the number of ticks consumed by the specified thread.

Note

This function is only available when the CH_DBG_THREADS_PROFILING configuration option is enabled.

Parameters

in	<i>tp</i>	pointer to the thread
----	-----------	-----------------------

Returns

The number of consumed system ticks.

Function Class:

This is an **X-Class** API, this function can be invoked from any context.

9.8.4.26 static bool chThdTerminatedX(thread_t * tp) [inline], [static]

Verifies if the specified thread is in the CH_STATE_FINAL state.

Parameters

in	<i>tp</i>	pointer to the thread
----	-----------	-----------------------

Return values

<i>true</i>	thread terminated.
<i>false</i>	thread not terminated.

Function Class:

This is an **X-Class** API, this function can be invoked from any context.

9.8.4.27 static bool chThdShouldTerminateX(void) [inline], [static]

Verifies if the current thread has a termination request pending.

Return values

<i>true</i>	termination request pending.
<i>false</i>	termination request not pending.

Function Class:

This is an **X-Class** API, this function can be invoked from any context.

Here is the call graph for this function:



9.8.4.28 static thread_t* chThdStartI(thread_t * tp) [inline], [static]

Resumes a thread created with [chThdCreateI\(\)](#).

Parameters

in	<i>tp</i>	pointer to the thread
----	-----------	-----------------------

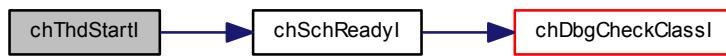
Returns

The pointer to the `thread_t` structure allocated for the thread into the working space area.

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:



9.8.4.29 static void chThdSleepS(systime_t time) [inline], [static]

Suspends the invoking thread for the specified time.

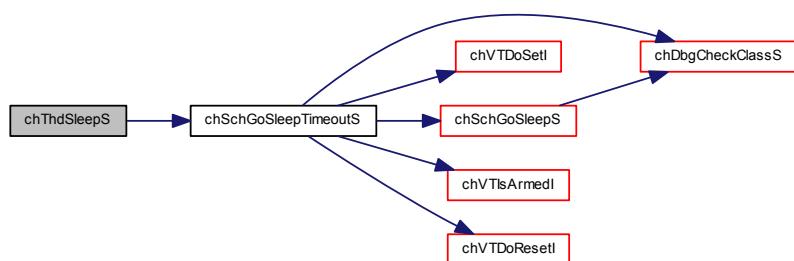
Parameters

in	<i>time</i>	the delay in system ticks, the special values are handled as follow: <ul style="list-style-type: none"> • <i>TIME_INFINITE</i> the thread enters an infinite sleep state. • <i>TIME_IMMEDIATE</i> this value is not allowed.
----	-------------	--

Function Class:

This is an **S-Class** API, this function can be invoked from within a system lock zone by threads only.

Here is the call graph for this function:



9.8.4.30 static void chThdQueueObjectInit(threads_queue_t * tqp) [inline], [static]

Initializes a threads queue object.

Parameters

out	<i>tqp</i>	pointer to the threads queue object
-----	------------	-------------------------------------

Function Class:

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

Here is the call graph for this function:



9.8.4.31 static bool chThdQueueIsEmpty(threads_queue_t * tqp) [inline], [static]

Evaluates to `true` if the specified queue is empty.

Parameters

out	<i>tqp</i>	pointer to the threads queue object
-----	------------	-------------------------------------

Returns

The queue status.

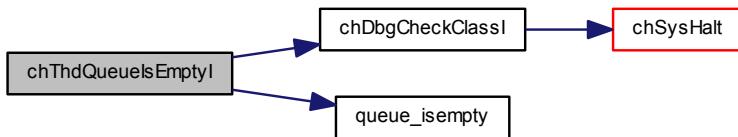
Return values

<code>false</code>	if the queue is not empty.
<code>true</code>	if the queue is empty.

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:



9.8.4.32 static void chThdDoDequeueNextI (threads_queue_t * tqp, msg_t msg) [inline], [static]

Dequeues and wakes up one thread from the threads queue object.

Dequeues one thread from the queue without checking if the queue is empty.

Precondition

The queue must contain at least an object.

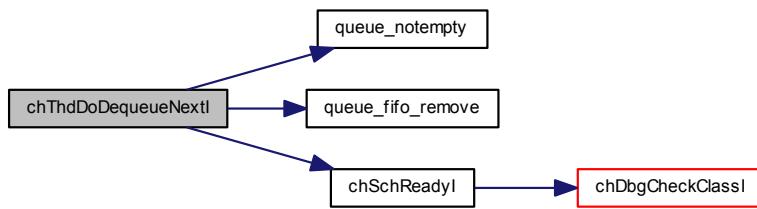
Parameters

in	<i>tqp</i>	pointer to the threads queue object
in	<i>msg</i>	the message code

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:



9.9 Time and Virtual Timers

9.9.1 Detailed Description

Time and Virtual Timers related APIs and services.

Special time constants

- `#define TIME_IMMEDIATE ((systime_t)0)`
Zero time specification for some functions with a timeout specification.
- `#define TIME_INFINITE ((systime_t)-1)`
Infinite time specification for all functions with a timeout specification.

Time conversion utilities

- `#define S2ST(sec) ((systime_t)((uint32_t)(sec) * (uint32_t)CH_CFG_ST_FREQUENCY))`
Seconds to system ticks.
- `#define MS2ST(msec)`
Milliseconds to system ticks.
- `#define US2ST(usec)`
Microseconds to system ticks.
- `#define ST2S(n) (((n) + CH_CFG_ST_FREQUENCY - 1UL) / CH_CFG_ST_FREQUENCY)`
System ticks to seconds.
- `#define ST2MS(n)`
System ticks to milliseconds.
- `#define ST2US(n)`
System ticks to microseconds.

Functions

- `void _vt_init (void)`
Virtual Timers initialization.
- `void chVTDoSetl (virtual_timer_t *vtp, systime_t delay, vtfunc_t vtfunc, void *par)`
Enables a virtual timer.
- `void chVTDoResetl (virtual_timer_t *vtp)`
Disables a Virtual Timer.
- `static void chVTOBJECTInit (virtual_timer_t *vtp)`
Initializes a virtual_timer_t object.
- `static systime_t chVTGetSystemTimeX (void)`
Current system time.
- `static systime_t chVTGetSystemTime (void)`
Current system time.
- `static systime_t chVTTIMEElapsedSinceX (systime_t start)`
Returns the elapsed time since the specified start time.
- `static bool chVTIsTimeWithinX (systime_t time, systime_t start, systime_t end)`
Checks if the specified time is within the specified time window.
- `static bool chVTIsSystemTimeWithinX (systime_t start, systime_t end)`
Checks if the current system time is within the specified time window.
- `static bool chVTIsSystemTimeWithin (systime_t start, systime_t end)`
Checks if the current system time is within the specified time window.

- static bool `chVTGetTimersStatel` (`systime_t` *`timep`)
Returns the time interval until the next timer event.
- static bool `chVTIsArmedl` (`virtual_timer_t` *`vtp`)
Returns true if the specified timer is armed.
- static bool `chVTIsArmed` (`virtual_timer_t` *`vtp`)
Returns true if the specified timer is armed.
- static void `chVTResetl` (`virtual_timer_t` *`vtp`)
Disables a Virtual Timer.
- static void `chVTReset` (`virtual_timer_t` *`vtp`)
Disables a Virtual Timer.
- static void `chVTSetl` (`virtual_timer_t` *`vtp`, `systime_t` `delay`, `vfunc_t` `vfunc`, `void` *`par`)
Enables a virtual timer.
- static void `chVTSet` (`virtual_timer_t` *`vtp`, `systime_t` `delay`, `vfunc_t` `vfunc`, `void` *`par`)
Enables a virtual timer.
- static void `chVTDoTickl` (`void`)
Virtual timers ticker.

9.9.2 Macro Definition Documentation

9.9.2.1 #define TIME_IMMEDIATE ((systime_t)0)

Zero time specification for some functions with a timeout specification.

Note

Not all functions accept `TIME_IMMEDIATE` as timeout parameter, see the specific function documentation.

9.9.2.2 #define TIME_INFINITE ((systime_t)-1)

Infinite time specification for all functions with a timeout specification.

Note

Not all functions accept `TIME_INFINITE` as timeout parameter, see the specific function documentation.

9.9.2.3 #define S2ST(sec) ((systime_t)((uint32_t)(sec) * (uint32_t)CH_CFG_ST_FREQUENCY))

Seconds to system ticks.

Converts from seconds to system ticks number.

Note

The result is rounded upward to the next tick boundary.

Parameters

in	sec	number of seconds
----	-----	-------------------

Returns

The number of ticks.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

9.9.2.4 #define MS2ST(msec)

Value:

```
((systime_t) (((((uint32_t)(msec)) * \
((uint32_t)CH_CFG_ST_FREQUENCY)) + 999UL) / 1000UL)) \
```

Milliseconds to system ticks.

Converts from milliseconds to system ticks number.

Note

The result is rounded upward to the next tick boundary.

Parameters

in	msec	number of milliseconds
----	------	------------------------

Returns

The number of ticks.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

9.9.2.5 #define US2ST(usec)

Value:

```
((systime_t) (((((uint32_t)(usec)) * \
((uint32_t)CH_CFG_ST_FREQUENCY)) + 999999UL) / 1000000UL)) \
```

Microseconds to system ticks.

Converts from microseconds to system ticks number.

Note

The result is rounded upward to the next tick boundary.

Parameters

in	usec	number of microseconds
----	------	------------------------

Returns

The number of ticks.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

9.9.2.6 #define ST2S(n) (((n) + CH_CFG_ST_FREQUENCY - 1UL) / CH_CFG_ST_FREQUENCY)

System ticks to seconds.

Converts from system ticks number to seconds.

Note

The result is rounded up to the next second boundary.

Parameters

in	<i>n</i>	number of system ticks
----	----------	------------------------

Returns

The number of seconds.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

9.9.2.7 #define ST2MS(*n*)**Value:**

```
(( (n) * 1000UL + CH_CFG_ST_FREQUENCY - 1UL) / \
```

System ticks to milliseconds.

Converts from system ticks number to milliseconds.

Note

The result is rounded up to the next millisecond boundary.

Parameters

in	<i>n</i>	number of system ticks
----	----------	------------------------

Returns

The number of milliseconds.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

9.9.2.8 #define ST2US(*n*)**Value:**

```
(( (n) * 1000000UL + CH_CFG_ST_FREQUENCY - 1UL) / \
```

System ticks to microseconds.

Converts from system ticks number to microseconds.

Note

The result is rounded up to the next microsecond boundary.

Parameters

in	<i>n</i>	number of system ticks
----	----------	------------------------

Returns

The number of microseconds.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

9.9.3 Function Documentation**9.9.3.1 void _vt_init(void)**

Virtual Timers initialization.

Note

Internal use only.

Function Class:

Not an API, this function is for internal use only.

9.9.3.2 void chVTDSetl(virtual_timer_t * vtp, systime_t delay, vtfunc_t vtfunc, void * par)

Enables a virtual timer.

The timer is enabled and programmed to trigger after the delay specified as parameter.

Precondition

The timer must not be already armed before calling this function.

Note

The callback function is invoked from interrupt context.

Parameters

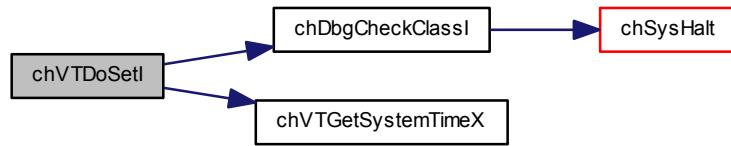
out	<i>vtp</i>	the <code>virtual_timer_t</code> structure pointer
in	<i>delay</i>	<p>the number of ticks before the operation timeouts, the special values are handled as follow:</p> <ul style="list-style-type: none"> • <code>TIME_INFINITE</code> is allowed but interpreted as a normal time specification. • <code>TIME_IMMEDIATE</code> this value is not allowed.

in	<i>vfunc</i>	the timer callback function. After invoking the callback the timer is disabled and the structure can be disposed or reused.
in	<i>par</i>	a parameter that will be passed to the callback function

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:

**9.9.3.3 void chVTDoResetl (virtual_timer_t * vtp)**

Disables a Virtual Timer.

Precondition

The timer must be in armed state before calling this function.

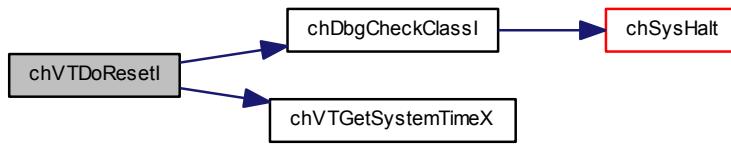
Parameters

in	<i>vtp</i>	the <code>virtual_timer_t</code> structure pointer
----	------------	--

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:



9.9.3.4 static void chVTOBJECTINIT(virtual_timer_t * vtp) [inline], [static]

Initializes a `virtual_timer_t` object.

Note

Initializing a timer object is not strictly required because the function `chVTSetI()` initializes the object too. This function is only useful if you need to perform a `chVTIsArmed()` check before calling `chVTSetI()`.

Parameters

out	<i>vtp</i>	the <code>virtual_timer_t</code> structure pointer
-----	------------	--

Function Class:

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

9.9.3.5 static systime_t chVTGetSystemTimeX(void) [inline], [static]

Current system time.

Returns the number of system ticks since the `chSysInit()` invocation.

Note

The counter can reach its maximum and then restart from zero.
This function can be called from any context but its atomicity is not guaranteed on architectures whose word size is less than `systime_t` size.

Returns

The system time in ticks.

Function Class:

This is an **X-Class** API, this function can be invoked from any context.

9.9.3.6 static systime_t chVTGetSystemTime(void) [inline], [static]

Current system time.

Returns the number of system ticks since the `chSysInit()` invocation.

Note

The counter can reach its maximum and then restart from zero.

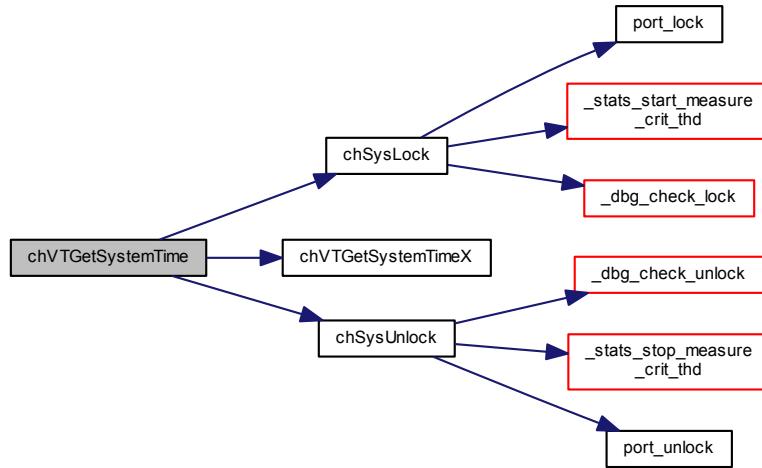
Returns

The system time in ticks.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



9.9.3.7 static systime_t chVTTIMEElapsedSinceX(systime_t start) [inline], [static]

Returns the elapsed time since the specified start time.

Parameters

in	<i>start</i>	start time
----	--------------	------------

Returns

The elapsed time.

Function Class:

This is an **X-Class** API, this function can be invoked from any context.

Here is the call graph for this function:



9.9.3.8 static bool chVTIsTimeWithinX(systime_t time, systime_t start, systime_t end) [inline], [static]

Checks if the specified time is within the specified time window.

Note

When start==end then the function returns always true because the whole time range is specified.
This function can be called from any context.

Parameters

in	<i>time</i>	the time to be verified
in	<i>start</i>	the start of the time window (inclusive)
in	<i>end</i>	the end of the time window (non inclusive)

Return values

<i>true</i>	current time within the specified time window.
<i>false</i>	current time not within the specified time window.

Function Class:

This is an **X-Class** API, this function can be invoked from any context.

9.9.3.9 static bool chVTIsSystemTimeWithinX (systime_t start, systime_t end) [inline], [static]

Checks if the current system time is within the specified time window.

Note

When start==end then the function returns always true because the whole time range is specified.

Parameters

in	<i>start</i>	the start of the time window (inclusive)
in	<i>end</i>	the end of the time window (non inclusive)

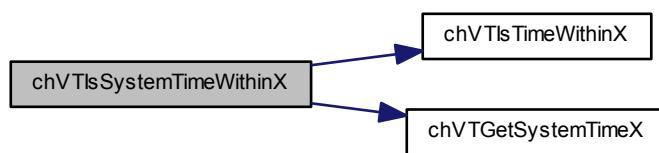
Return values

<i>true</i>	current time within the specified time window.
<i>false</i>	current time not within the specified time window.

Function Class:

This is an **X-Class** API, this function can be invoked from any context.

Here is the call graph for this function:



9.9.3.10 static bool chVTIsSystemTimeWithin (*systime_t start, systime_t end*) [inline], [static]

Checks if the current system time is within the specified time window.

Note

When *start==end* then the function returns always true because the whole time range is specified.

Parameters

in	<i>start</i>	the start of the time window (inclusive)
in	<i>end</i>	the end of the time window (non inclusive)

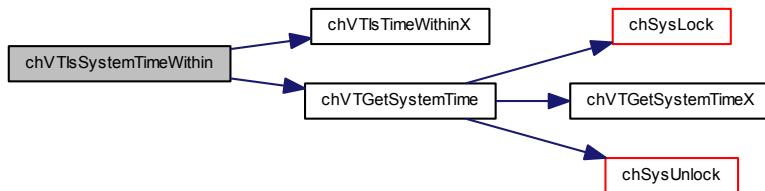
Return values

<i>true</i>	current time within the specified time window.
<i>false</i>	current time not within the specified time window.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

9.9.3.11 static bool chVTGetTimersState (*systime_t * timep*) [inline], [static]

Returns the time interval until the next timer event.

Note

The return value is not perfectly accurate and can report values in excess of CH_CFG_ST_TIMEDELTA ticks.

The interval returned by this function is only meaningful if more timers are not added to the list until the returned time.

Parameters

out	<i>timep</i>	pointer to a variable that will contain the time interval until the next timer elapses. This pointer can be <code>NULL</code> if the information is not required.
-----	--------------	---

Returns

The time, in ticks, until next time event.

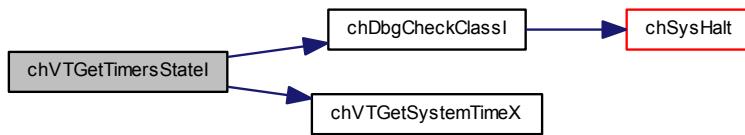
Return values

<i>false</i>	if the timers list is empty.
<i>true</i>	if the timers list contains at least one timer.

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:

**9.9.3.12 static bool chVTIsArmed(virtual_timer_t * vtp) [inline], [static]**

Returns *true* if the specified timer is armed.

Precondition

The timer must have been initialized using [chVTOBJECTINIT\(\)](#) or [chVTDOSETI\(\)](#).

Parameters

in	<i>vtp</i>	the <code>virtual_timer_t</code> structure pointer
----	------------	--

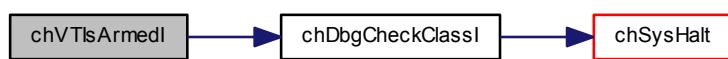
Returns

true if the timer is armed.

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:



9.9.3.13 static bool chVTIsArmed (virtual_timer_t * vtp) [inline], [static]

Returns `true` if the specified timer is armed.

Precondition

The timer must have been initialized using `chVTOBJECTINIT()` or `chVTDOSETI()`.

Parameters

in	<code>vtp</code>	the <code>virtual_timer_t</code> structure pointer
----	------------------	--

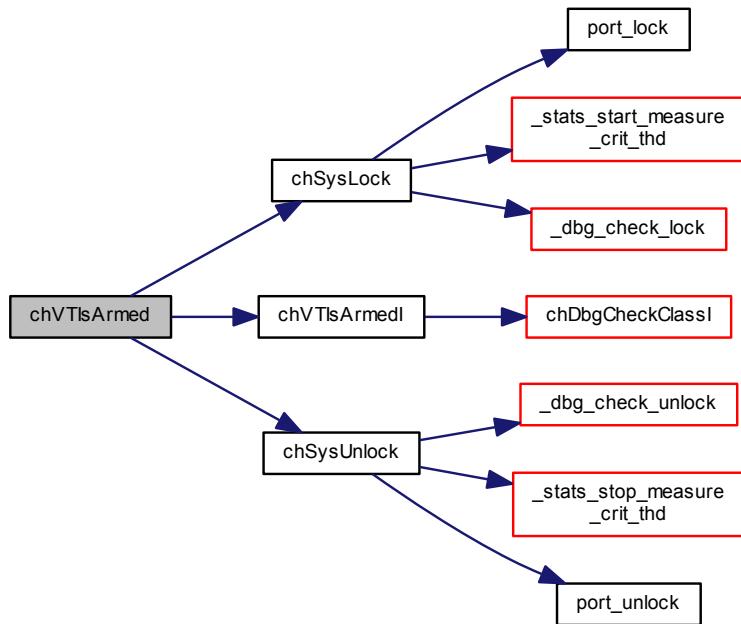
Returns

`true` if the timer is armed.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



9.9.3.14 static void chVTResetI (virtual_timer_t * vtp) [inline], [static]

Disables a Virtual Timer.

Note

The timer is first checked and disabled only if armed.

Precondition

The timer must have been initialized using `chVTOBJECTINIT()` or `chVTDOSETI()`.

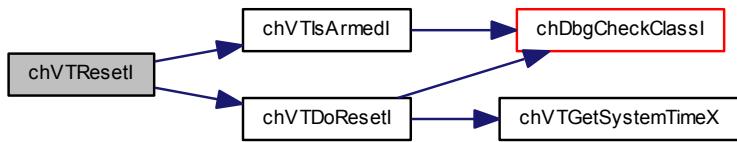
Parameters

in	<code>vtp</code>	the <code>virtual_timer_t</code> structure pointer
----	------------------	--

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:



9.9.3.15 static void chVTReset(virtual_timer_t * vtp) [inline], [static]

Disables a Virtual Timer.

Note

The timer is first checked and disabled only if armed.

Precondition

The timer must have been initialized using `chVTObjectInit()` or `chVTDosetI()`.

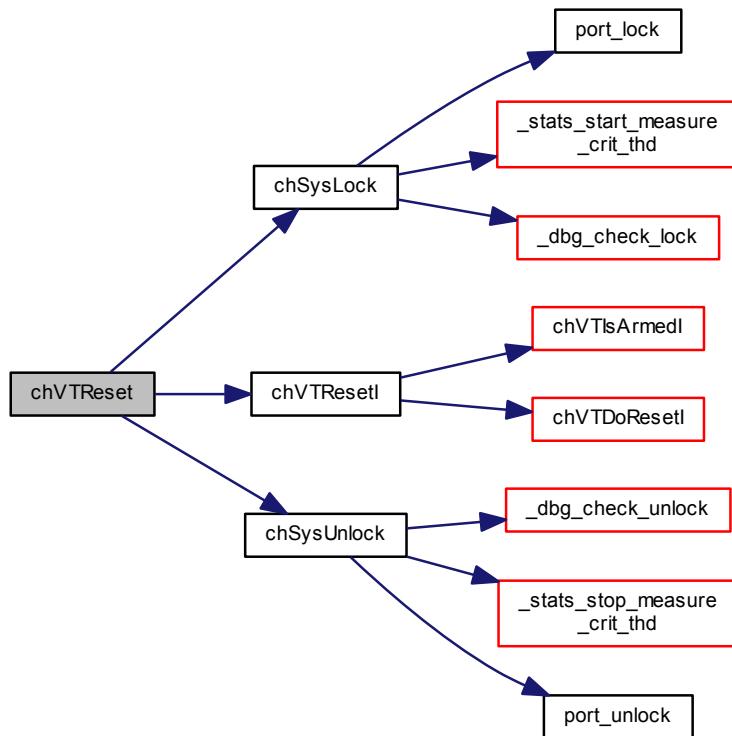
Parameters

in *vtp* the virtual_timer_t structure pointer

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



9.9.3.16 static void chVTSetl (virtual_timer_t *vip, systime_t delay, vtfunc_t vtfunc, void *par) [inline],
[static]

Enables a virtual timer.

If the virtual timer was already enabled then it is re-enabled using the new parameters.

Precondition

The timer must have been initialized using `chVTOBJECTINIT()` or `chVTDOSETI()`.

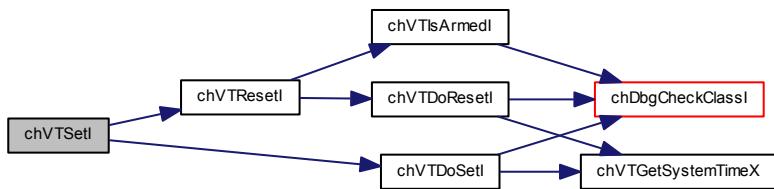
Parameters

in	<i>vtp</i>	the <code>virtual_timer_t</code> structure pointer
in	<i>delay</i>	the number of ticks before the operation timeouts, the special values are handled as follow: <ul style="list-style-type: none">• <code>TIME_INFINITE</code> is allowed but interpreted as a normal time specification.• <code>TIME_IMMEDIATE</code> this value is not allowed.
in	<i>vfunc</i>	the timer callback function. After invoking the callback the timer is disabled and the structure can be disposed or reused.
in	<i>par</i>	a parameter that will be passed to the callback function

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:



9.9.3.17 static void chVTSet (virtual_timer_t * *vtp*, systime_t *delay*, vfunc_t *vfunc*, void * *par*) [inline], [static]

Enables a virtual timer.

If the virtual timer was already enabled then it is re-enabled using the new parameters.

Precondition

The timer must have been initialized using `chVTObjectInit()` or `chVTDoSetI()`.

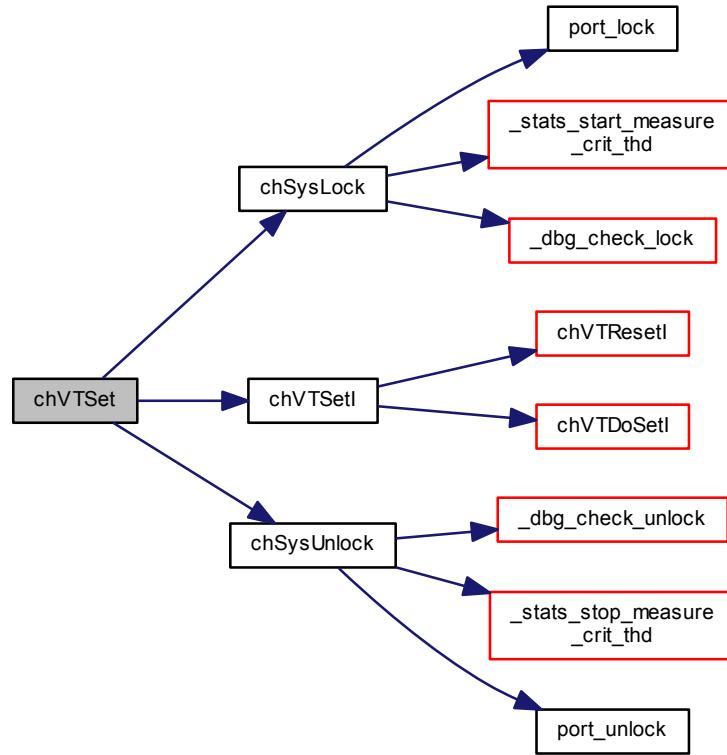
Parameters

in	<i>vtp</i>	the <code>virtual_timer_t</code> structure pointer
in	<i>delay</i>	the number of ticks before the operation timeouts, the special values are handled as follow: <ul style="list-style-type: none">• <code>TIME_INFINITE</code> is allowed but interpreted as a normal time specification.• <code>TIME_IMMEDIATE</code> this value is not allowed.
in	<i>vfunc</i>	the timer callback function. After invoking the callback the timer is disabled and the structure can be disposed or reused.
in	<i>par</i>	a parameter that will be passed to the callback function

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

**9.9.3.18 static void chVTDoTickl (void) [inline], [static]**

Virtual timers ticker.

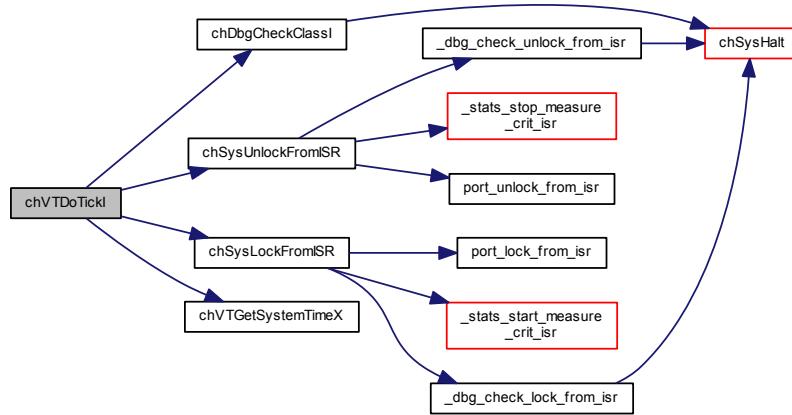
Note

The system lock is released before entering the callback and re-acquired immediately after. It is callback's responsibility to acquire the lock if needed. This is done in order to reduce interrupts jitter when many timers are in use.

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:



9.10 Synchronization

9.10.1 Detailed Description

Synchronization services.

Modules

- [Counting Semaphores](#)
- [Binary Semaphores](#)
- [Mutexes](#)
- [Condition Variables](#)
- [Event Flags](#)
- [Synchronous Messages](#)
- [Mailboxes](#)
- [I/O Queues](#)

9.11 Counting Semaphores

9.11.1 Detailed Description

Semaphores related APIs and services.

Operation mode

Semaphores are a flexible synchronization primitive, ChibiOS/RT implements semaphores in their "counting semaphores" variant as defined by Edsger Dijkstra plus several enhancements like:

- Wait operation with timeout.
- Reset operation.
- Atomic wait+signal operation.
- Return message from the wait operation (OK, RESET, TIMEOUT).

The binary semaphores variant can be easily implemented using counting semaphores.

Operations defined for semaphores:

- **Signal:** The semaphore counter is increased and if the result is non-positive then a waiting thread is removed from the semaphore queue and made ready for execution.
- **Wait:** The semaphore counter is decreased and if the result becomes negative the thread is queued in the semaphore and suspended.
- **Reset:** The semaphore counter is reset to a non-negative value and all the threads in the queue are released.

Semaphores can be used as guards for mutual exclusion zones (note that mutexes are recommended for this kind of use) but also have other uses, queues guards and counters for example.

Semaphores usually use a FIFO queuing strategy but it is possible to make them order threads by priority by enabling CH_CFG_USE_SEMAPHORES_PRIORITY in [chconf.h](#).

Precondition

In order to use the semaphore APIs the CH_CFG_USE_SEMAPHORES option must be enabled in [chconf.h](#).

Macros

- #define [_SEMAPHORE_DATA](#)(name, n) {[_THREADS_QUEUE_DATA](#)(name.s_queue), n}
Data part of a static semaphore initializer.
- #define [SEMAPHORE_DECL](#)(name, n) [semaphore_t](#) name = [_SEMAPHORE_DATA](#)(name, n)
Static semaphore initializer.

TypeDefs

- typedef struct [ch_semaphore](#) [semaphore_t](#)
Semaphore structure.

Data Structures

- struct [ch_semaphore](#)
Semaphore structure.

Functions

- `void chSemObjectInit (semaphore_t *sp, cnt_t n)`
Initializes a semaphore with the specified counter value.
- `void chSemReset (semaphore_t *sp, cnt_t n)`
Performs a reset operation on the semaphore.
- `void chSemResetl (semaphore_t *sp, cnt_t n)`
Performs a reset operation on the semaphore.
- `msg_t chSemWait (semaphore_t *sp)`
Performs a wait operation on a semaphore.
- `msg_t chSemWaitS (semaphore_t *sp)`
Performs a wait operation on a semaphore.
- `msg_t chSemWaitTimeout (semaphore_t *sp, systime_t time)`
Performs a wait operation on a semaphore with timeout specification.
- `msg_t chSemWaitTimeoutS (semaphore_t *sp, systime_t time)`
Performs a wait operation on a semaphore with timeout specification.
- `void chSemSignal (semaphore_t *sp)`
Performs a signal operation on a semaphore.
- `void chSemSignall (semaphore_t *sp)`
Performs a signal operation on a semaphore.
- `void chSemAddCounterl (semaphore_t *sp, cnt_t n)`
Adds the specified value to the semaphore counter.
- `msg_t chSemSignalWait (semaphore_t *sp, semaphore_t *spw)`
Performs atomic signal and wait operations on two semaphores.
- `static void chSemFastWaitl (semaphore_t *sp)`
Decreases the semaphore counter.
- `static void chSemFastSignall (semaphore_t *sp)`
Increases the semaphore counter.
- `static cnt_t chSemGetCounterl (semaphore_t *sp)`
Returns the semaphore counter current value.

9.11.2 Macro Definition Documentation

9.11.2.1 `#define _SEMAPHORE_DATA(name, n) { _THREADS_QUEUE_DATA(name.s_queue), n }`

Data part of a static semaphore initializer.

This macro should be used when statically initializing a semaphore that is part of a bigger structure.

Parameters

in	name	the name of the semaphore variable
in	n	the counter initial value, this value must be non-negative

9.11.2.2 `#define SEMAPHORE_DECL(name, n) semaphore_t name = _SEMAPHORE_DATA(name, n)`

Static semaphore initializer.

Statically initialized semaphores require no explicit initialization using `chSemInit ()`.

Parameters

in	<i>name</i>	the name of the semaphore variable
in	<i>n</i>	the counter initial value, this value must be non-negative

9.11.3 TYPEDOC Documentation**9.11.3.1 `typedef struct ch_semaphore semaphore_t`**

Semaphore structure.

9.11.4 FUNCTION Documentation**9.11.4.1 `void chSemObjectInit(semaphore_t * sp, cnt_t n)`**

Initializes a semaphore with the specified counter value.

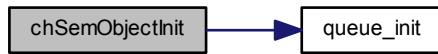
Parameters

out	<i>sp</i>	pointer to a <code>semaphore_t</code> structure
in	<i>n</i>	initial value of the semaphore counter. Must be non-negative.

Function Class:

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

Here is the call graph for this function:

**9.11.4.2 `void chSemReset(semaphore_t * sp, cnt_t n)`**

Performs a reset operation on the semaphore.

Postcondition

After invoking this function all the threads waiting on the semaphore, if any, are released and the semaphore counter is set to the specified, non negative, value.

Note

The released threads can recognize they were waken up by a reset rather than a signal because the `ch<- SemWait()` will return `MSG_RESET` instead of `MSG_OK`.

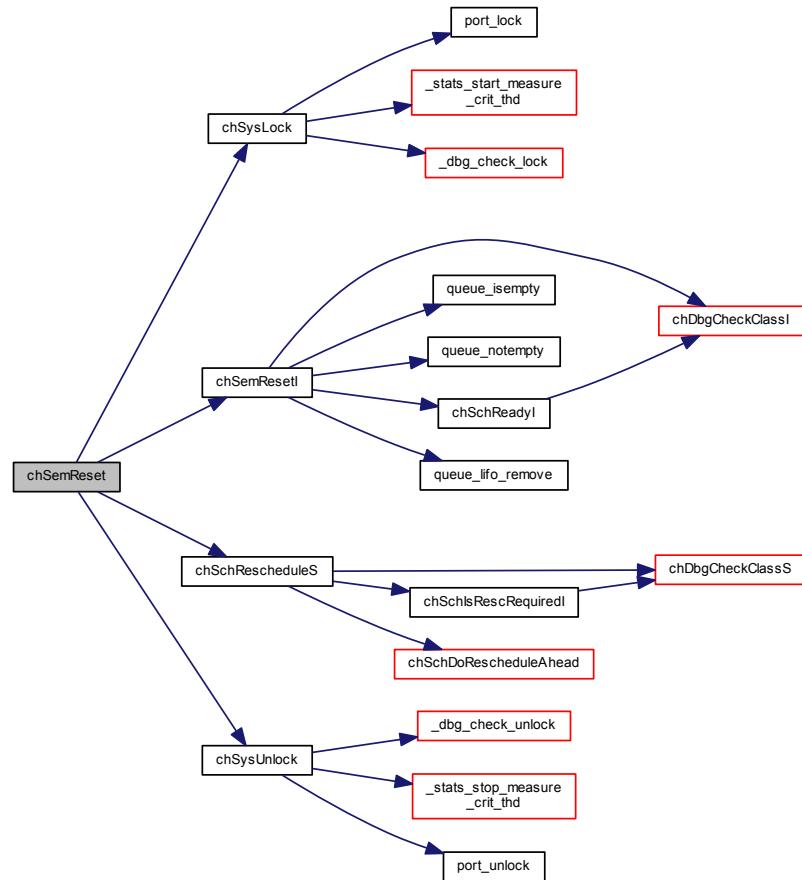
Parameters

in	<i>sp</i>	pointer to a <code>semaphore_t</code> structure
in	<i>n</i>	the new value of the semaphore counter. The value must be non-negative.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



9.11.4.3 void chSemReset(semaphore_t * sp, cnt_t n)

Performs a reset operation on the semaphore.

Postcondition

After invoking this function all the threads waiting on the semaphore, if any, are released and the semaphore counter is set to the specified, non negative, value.

This function does not reschedule so a call to a rescheduling function must be performed before unlocking the kernel. Note that interrupt handlers always reschedule on exit so an explicit reschedule must not be performed in ISRs.

Note

The released threads can recognize they were wakened up by a reset rather than a signal because the `chSemWait()` will return `MSG_RESET` instead of `MSG_OK`.

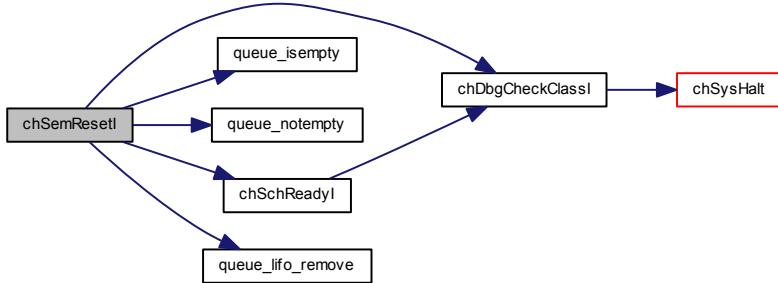
Parameters

in	<i>sp</i>	pointer to a <code>semaphore_t</code> structure
in	<i>n</i>	the new value of the semaphore counter. The value must be non-negative.

Function Class:

This is an **I-Class API**, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:

**9.11.4.4 msg_t chSemWait(semaphore_t * sp)**

Performs a wait operation on a semaphore.

Parameters

in	<i>sp</i>	pointer to a <code>semaphore_t</code> structure
----	-----------	---

Returns

A message specifying how the invoking thread has been released from the semaphore.

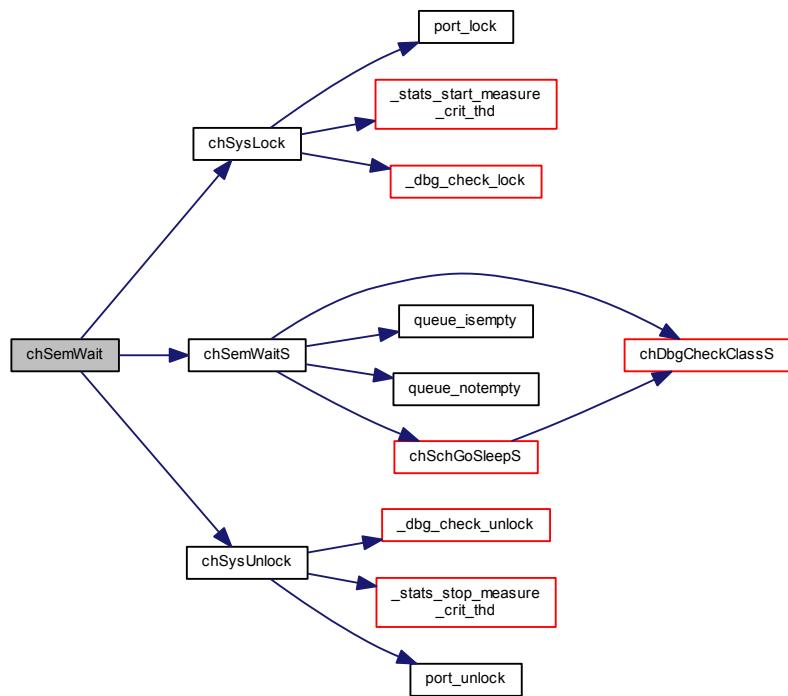
Return values

<code>MSG_OK</code>	if the thread has not stopped on the semaphore or the semaphore has been signaled.
<code>MSG_RESET</code>	if the semaphore has been reset using <code>chSemReset()</code> .

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



9.11.4.5 `msg_t chSemWaitS(semaphore_t * sp)`

Performs a wait operation on a semaphore.

Parameters

in	<code>sp</code>	pointer to a <code>semaphore_t</code> structure
----	-----------------	---

Returns

A message specifying how the invoking thread has been released from the semaphore.

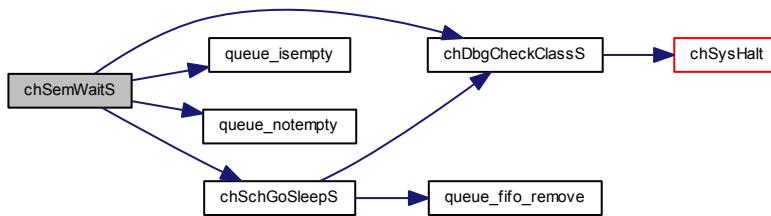
Return values

<i>MSG_OK</i>	if the thread has not stopped on the semaphore or the semaphore has been signaled.
<i>MSG_RESET</i>	if the semaphore has been reset using chSemReset () .

Function Class:

This is an **S-Class API**, this function can be invoked from within a system lock zone by threads only.

Here is the call graph for this function:

**9.11.4.6 msg_t chSemWaitTimeout (semaphore_t * sp, systime_t time)**

Performs a wait operation on a semaphore with timeout specification.

Parameters

in	<i>sp</i>	pointer to a <code>semaphore_t</code> structure
in	<i>time</i>	the number of ticks before the operation timeouts, the following special values are allowed: <ul style="list-style-type: none"> • <code>TIME_IMMEDIATE</code> immediate timeout. • <code>TIME_INFINITE</code> no timeout.

Returns

A message specifying how the invoking thread has been released from the semaphore.

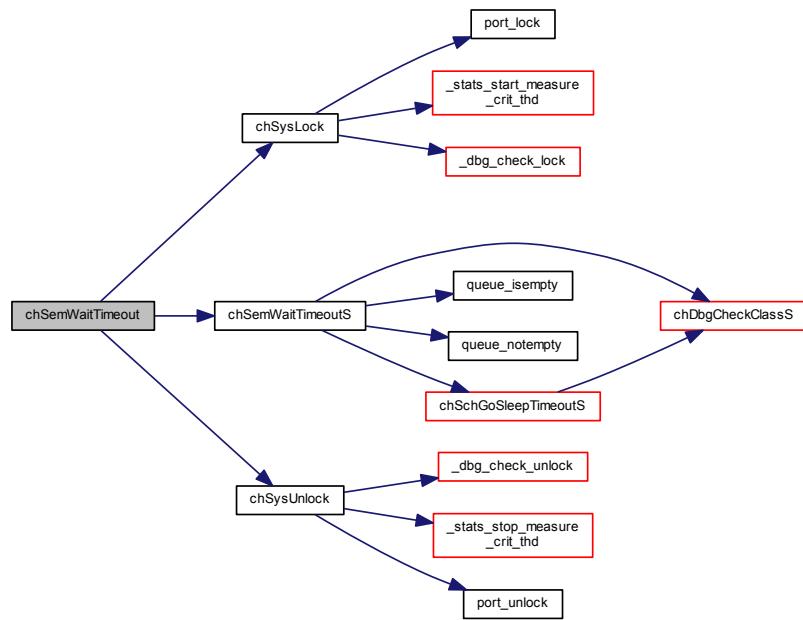
Return values

<i>MSG_OK</i>	if the thread has not stopped on the semaphore or the semaphore has been signaled.
<i>MSG_RESET</i>	if the semaphore has been reset using chSemReset () .
<i>MSG_TIMEOUT</i>	if the semaphore has not been signaled or reset within the specified timeout.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



9.11.4.7 msg_t chSemWaitTimeoutS (semaphore_t * sp, systime_t time)

Performs a wait operation on a semaphore with timeout specification.

Parameters

in	<i>sp</i>	pointer to a <code>semaphore_t</code> structure
in	<i>time</i>	the number of ticks before the operation timeouts, the following special values are allowed: <ul style="list-style-type: none"> • <code>TIME_IMMEDIATE</code> immediate timeout. • <code>TIME_INFINITE</code> no timeout.

Returns

A message specifying how the invoking thread has been released from the semaphore.

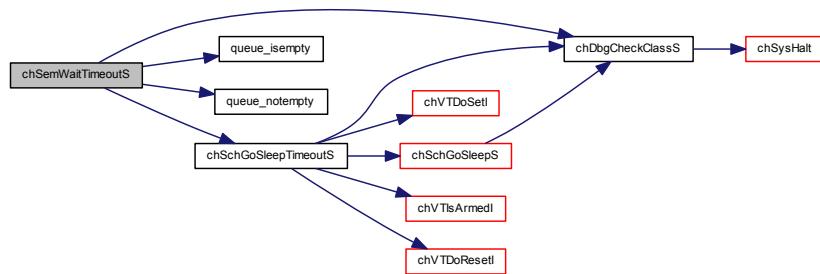
Return values

<code>MSG_OK</code>	if the thread has not stopped on the semaphore or the semaphore has been signaled.
<code>MSG_RESET</code>	if the semaphore has been reset using <code>chSemReset()</code> .
<code>MSG_TIMEOUT</code>	if the semaphore has not been signaled or reset within the specified timeout.

Function Class:

This is an **S-Class** API, this function can be invoked from within a system lock zone by threads only.

Here is the call graph for this function:



9.11.4.8 void chSemSignal (semaphore_t * sp)

Performs a signal operation on a semaphore.

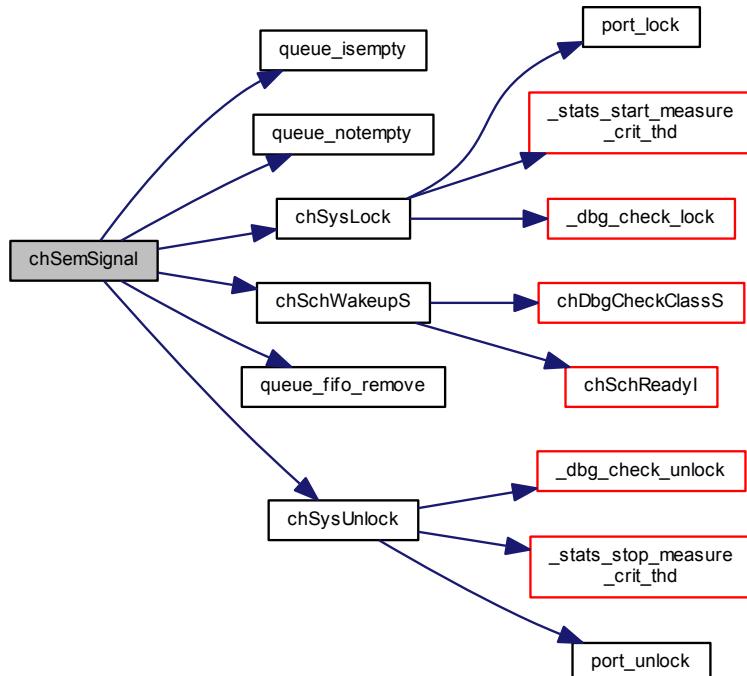
Parameters

in	<i>sp</i>	pointer to a <code>semaphore_t</code> structure
----	-----------	---

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



9.11.4.9 void chSemSignall (semaphore_t * sp)

Performs a signal operation on a semaphore.

Postcondition

This function does not reschedule so a call to a rescheduling function must be performed before unlocking the kernel. Note that interrupt handlers always reschedule on exit so an explicit reschedule must not be performed in ISRs.

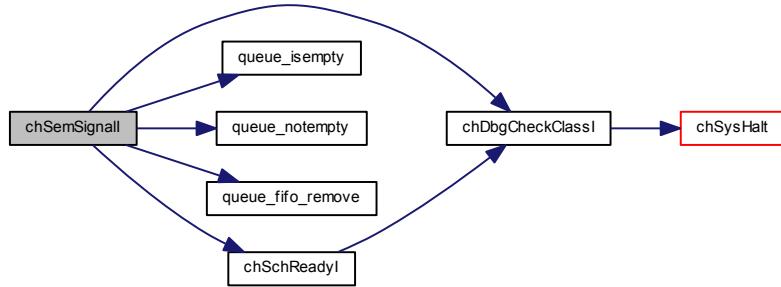
Parameters

in	sp	pointer to a semaphore_t structure
----	----	------------------------------------

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:



9.11.4.10 void chSemAddCounterI (semaphore_t * sp, cnt_t n)

Adds the specified value to the semaphore counter.

Postcondition

This function does not reschedule so a call to a rescheduling function must be performed before unlocking the kernel. Note that interrupt handlers always reschedule on exit so an explicit reschedule must not be performed in ISRs.

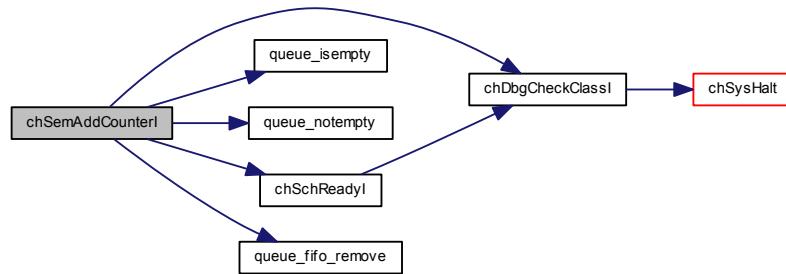
Parameters

in	sp	pointer to a semaphore_t structure
in	n	value to be added to the semaphore counter. The value must be positive.

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:



9.11.4.11 `msg_t chSemSignalWait(semaphore_t *sps, semaphore_t *spw)`

Performs atomic signal and wait operations on two semaphores.

Parameters

in	<code>sps</code>	pointer to a <code>semaphore_t</code> structure to be signaled
in	<code>spw</code>	pointer to a <code>semaphore_t</code> structure to wait on

Returns

A message specifying how the invoking thread has been released from the semaphore.

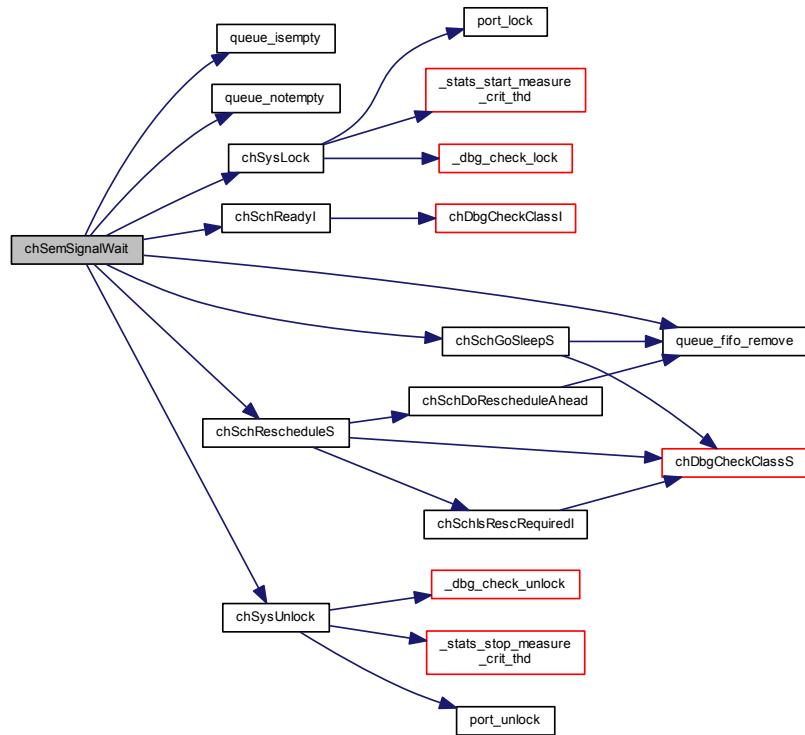
Return values

<code>MSG_OK</code>	if the thread has not stopped on the semaphore or the semaphore has been signaled.
<code>MSG_RESET</code>	if the semaphore has been reset using <code>chSemReset()</code> .

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



9.11.4.12 static void chSemFastWaitl(semaphore_t * sp) [inline], [static]

Decreases the semaphore counter.

This macro can be used when the counter is known to be positive.

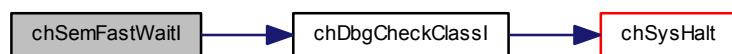
Parameters

in	sp	pointer to a semaphore_t structure
----	----	------------------------------------

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:



9.11.4.13 static void chSemFastSignall(semaphore_t * sp) [inline], [static]

Increases the semaphore counter.

This macro can be used when the counter is known to be not negative.

Parameters

in	<i>sp</i>	pointer to a <code>semaphore_t</code> structure
----	-----------	---

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:



9.11.4.14 static cnt_t chSemGetCounterl(semaphore_t * sp) [inline], [static]

Returns the semaphore counter current value.

Parameters

in	<i>sp</i>	pointer to a <code>semaphore_t</code> structure
----	-----------	---

Returns

The semaphore counter value.

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:



9.12 Binary Semaphores

9.12.1 Detailed Description

Binary semaphores related APIs and services.

Operation mode

Binary semaphores are implemented as a set of inline functions that use the existing counting semaphores primitives. The difference between counting and binary semaphores is that the counter of binary semaphores is not allowed to grow above the value 1. Repeated signal operation are ignored. A binary semaphore can thus have only two defined states:

- **Taken**, when its counter has a value of zero or lower than zero. A negative number represent the number of threads queued on the binary semaphore.
- **Not taken**, when its counter has a value of one.

Binary semaphores are different from mutexes because there is no concept of ownership, a binary semaphore can be taken by a thread and signaled by another thread or an interrupt handler, mutexes can only be taken and released by the same thread. Another difference is that binary semaphores, unlike mutexes, do not implement the priority inheritance protocol.

In order to use the binary semaphores APIs the `CH_CFG_USE_SEMAPHORES` option must be enabled in `chconf.h`.

Macros

- `#define _BSEMAPHORE_DATA(name, taken) { _SEMAPHORE_DATA(name.bs_sem, ((taken) ? 0 : 1))}`
Data part of a static semaphore initializer.
- `#define BSEMAPHORE_DECL(name, taken) binary_semaphore_t name = _BSEMAPHORE_DATA(name, taken)`
Static semaphore initializer.

Data Structures

- struct `binary_semaphore_t`
Binary semaphore type.

Functions

- static void `chBSemObjectInit (binary_semaphore_t *bsp, bool taken)`
Initializes a binary semaphore.
- static `msg_t chBSemWait (binary_semaphore_t *bsp)`
Wait operation on the binary semaphore.
- static `msg_t chBSemWaitS (binary_semaphore_t *bsp)`
Wait operation on the binary semaphore.
- static `msg_t chBSemWaitTimeoutS (binary_semaphore_t *bsp, systime_t time)`
Wait operation on the binary semaphore.
- static `msg_t chBSemWaitTimeout (binary_semaphore_t *bsp, systime_t time)`
Wait operation on the binary semaphore.
- static void `chBSemReset (binary_semaphore_t *bsp, bool taken)`
Reset operation on the binary semaphore.

- static void `chBSemReset (binary_semaphore_t *bsp, bool taken)`
Reset operation on the binary semaphore.
- static void `chBSemSignall (binary_semaphore_t *bsp)`
Performs a signal operation on a binary semaphore.
- static void `chBSemSignal (binary_semaphore_t *bsp)`
Performs a signal operation on a binary semaphore.
- static bool `chBSemGetStatel (binary_semaphore_t *bsp)`
Returns the binary semaphore current state.

9.12.2 Macro Definition Documentation

9.12.2.1 `#define _BSEMAPHORE_DATA(name, taken) {_SEMAPHORE_DATA(name.bs_sem, ((taken) ? 0 : 1))}`

Data part of a static semaphore initializer.

This macro should be used when statically initializing a semaphore that is part of a bigger structure.

Parameters

in	<i>name</i>	the name of the semaphore variable
in	<i>taken</i>	the semaphore initial state

9.12.2.2 `#define BSEMAPHORE_DECL(name, taken) binary_semaphore_t name = _BSEMAPHORE_DATA(name, taken)`

Static semaphore initializer.

Statically initialized semaphores require no explicit initialization using `chBSemInit ()`.

Parameters

in	<i>name</i>	the name of the semaphore variable
in	<i>taken</i>	the semaphore initial state

9.12.3 Function Documentation

9.12.3.1 `static void chBSemObjectInit (binary_semaphore_t * bsp, bool taken) [inline], [static]`

Initializes a binary semaphore.

Parameters

out	<i>bsp</i>	pointer to a <code>binary_semaphore_t</code> structure
in	<i>taken</i>	initial state of the binary semaphore: <ul style="list-style-type: none"> • <i>false</i>, the initial state is not taken. • <i>true</i>, the initial state is taken.

Function Class:

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

Here is the call graph for this function:



9.12.3.2 static msg_t chBSemWait(binary_semaphore_t * bsp) [inline], [static]

Wait operation on the binary semaphore.

Parameters

in	<i>bsp</i>	pointer to a binary_semaphore_t structure
----	------------	---

Returns

A message specifying how the invoking thread has been released from the semaphore.

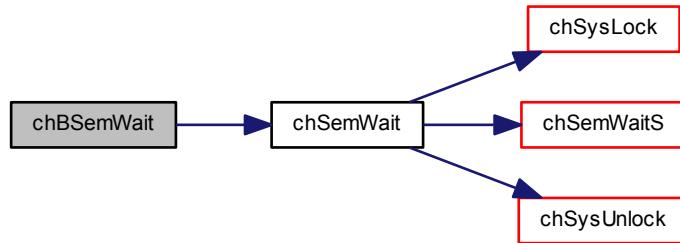
Return values

<i>MSG_OK</i>	if the binary semaphore has been successfully taken.
<i>MSG_RESET</i>	if the binary semaphore has been reset using <code>bsemReset()</code> .

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



9.12.3.3 static msg_t chBSemWaitS(binary_semaphore_t * bsp) [inline], [static]

Wait operation on the binary semaphore.

Parameters

in	<i>bsp</i>	pointer to a binary_semaphore_t structure
----	------------	---

Returns

A message specifying how the invoking thread has been released from the semaphore.

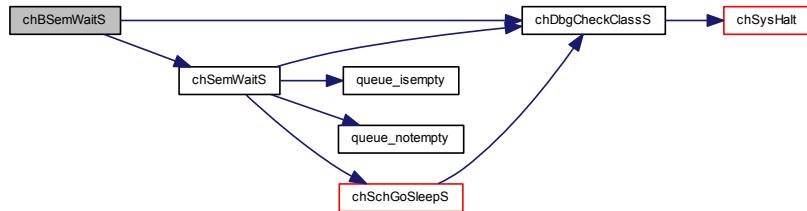
Return values

<i>MSG_OK</i>	if the binary semaphore has been successfully taken.
<i>MSG_RESET</i>	if the binary semaphore has been reset using <code>bsemReset()</code> .

Function Class:

This is an **S-Class** API, this function can be invoked from within a system lock zone by threads only.

Here is the call graph for this function:



9.12.3.4 static msg_t chBSemWaitTimeoutS (*binary_semaphore_t* * *bsp*, *systime_t* *time*) [inline], [static]

Wait operation on the binary semaphore.

Parameters

in	<i>bsp</i>	pointer to a binary_semaphore_t structure
in	<i>time</i>	the number of ticks before the operation timeouts, the following special values are allowed: <ul style="list-style-type: none"> • <i>TIME_IMMEDIATE</i> immediate timeout. • <i>TIME_INFINITE</i> no timeout.

Returns

A message specifying how the invoking thread has been released from the semaphore.

Return values

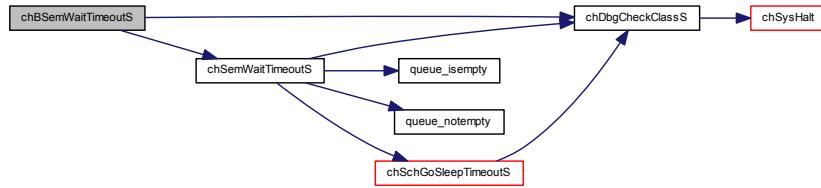
<i>MSG_OK</i>	if the binary semaphore has been successfully taken.
---------------	--

<i>MSG_RESET</i>	if the binary semaphore has been reset using <code>bsemReset()</code> .
<i>MSG_TIMEOUT</i>	if the binary semaphore has not been signaled or reset within the specified timeout.

Function Class:

This is an **S-Class** API, this function can be invoked from within a system lock zone by threads only.

Here is the call graph for this function:



9.12.3.5 static msg_t chBSemWaitTimeout (binary_semaphore_t * *bsp*, systime_t *time*) [inline], [static]

Wait operation on the binary semaphore.

Parameters

in	<i>bsp</i>	pointer to a <code>binary_semaphore_t</code> structure
in	<i>time</i>	the number of ticks before the operation timeouts, the following special values are allowed: <ul style="list-style-type: none"> • <code>TIME_IMMEDIATE</code> immediate timeout. • <code>TIME_INFINITE</code> no timeout.

Returns

A message specifying how the invoking thread has been released from the semaphore.

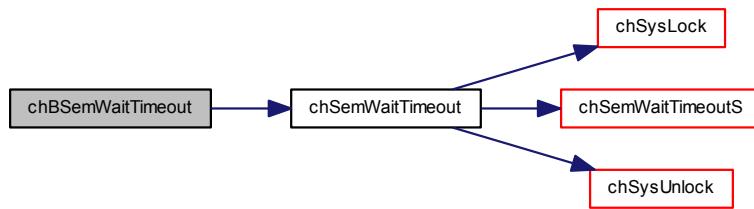
Return values

<i>MSG_OK</i>	if the binary semaphore has been successfully taken.
<i>MSG_RESET</i>	if the binary semaphore has been reset using <code>bsemReset()</code> .
<i>MSG_TIMEOUT</i>	if the binary semaphore has not been signaled or reset within the specified timeout.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



9.12.3.6 static void chBSemResetl ([binary_semaphore_t](#) * bsp, bool taken) [inline], [static]

Reset operation on the binary semaphore.

Note

The released threads can recognize they were waked up by a reset rather than a signal because the `bsem->Wait()` will return `MSG_RESET` instead of `MSG_OK`.

This function does not reschedule.

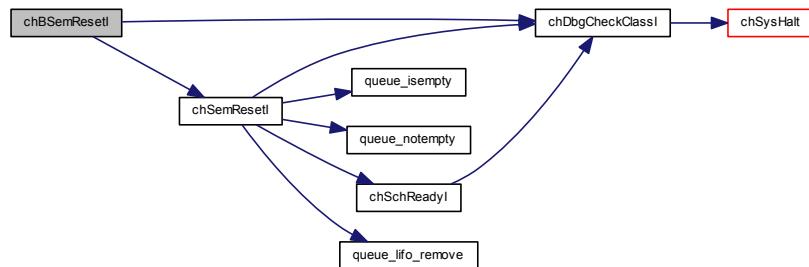
Parameters

in	<code>bsp</code>	pointer to a binary_semaphore_t structure
in	<code>taken</code>	new state of the binary semaphore <ul style="list-style-type: none"> • <i>false</i>, the new state is not taken. • <i>true</i>, the new state is taken.

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:



9.12.3.7 static void chBSemReset(*binary_semaphore_t* * *bsp*, *bool* *taken*) [inline], [static]

Reset operation on the binary semaphore.

Note

The released threads can recognize they were waked up by a reset rather than a signal because the `bsem->Wait()` will return `MSG_RESET` instead of `MSG_OK`.

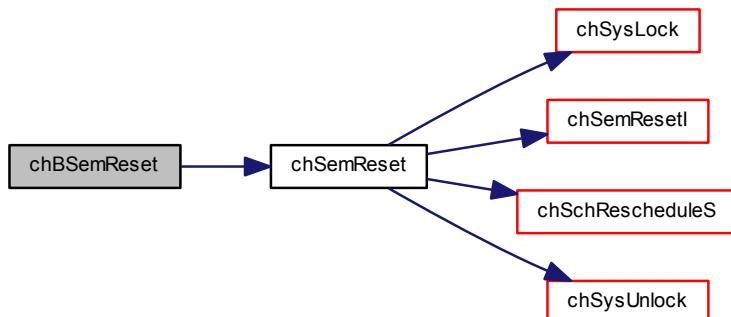
Parameters

in	<i>bsp</i>	pointer to a <code>binary_semaphore_t</code> structure
in	<i>taken</i>	new state of the binary semaphore <ul style="list-style-type: none"> • <i>false</i>, the new state is not taken. • <i>true</i>, the new state is taken.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

9.12.3.8 static void chBSemSignall(*binary_semaphore_t* * *bsp*) [inline], [static]

Performs a signal operation on a binary semaphore.

Note

This function does not reschedule.

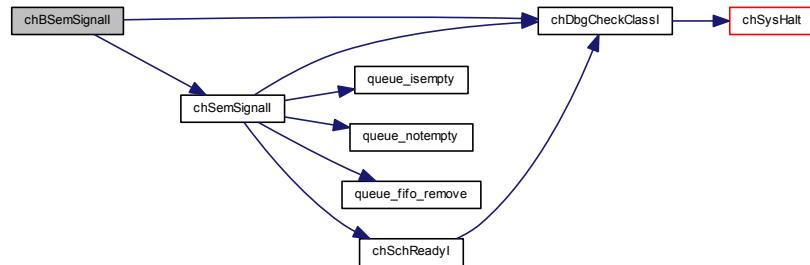
Parameters

in	<i>bsp</i>	pointer to a <code>binary_semaphore_t</code> structure
----	------------	--

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:



9.12.3.9 static void chBSemSignal (binary_semaphore_t * bsp) [inline], [static]

Performs a signal operation on a binary semaphore.

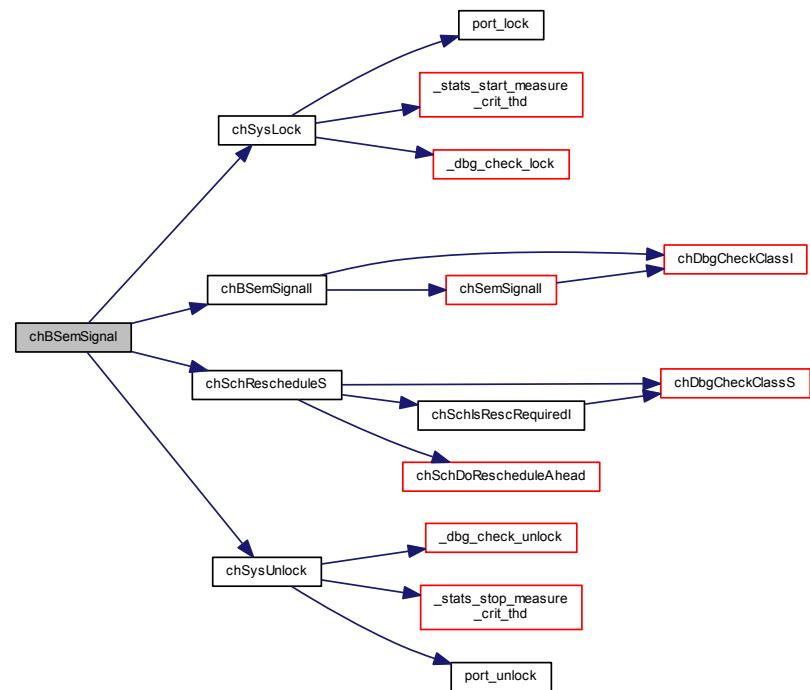
Parameters

in	<i>bsp</i>	pointer to a binary_semaphore_t structure
----	------------	---

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



9.12.3.10 static bool chBSemGetState(**binary_semaphore_t** * *bsp*) [inline], [static]

Returns the binary semaphore current state.

Parameters

in	<i>bsp</i>	pointer to a binary_semaphore_t structure
----	------------	---

Returns

The binary semaphore current state.

Return values

<i>false</i>	if the binary semaphore is not taken.
<i>true</i>	if the binary semaphore is taken.

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:



9.13 Mutexes

9.13.1 Detailed Description

Mutexes related APIs and services.

Operation mode

A mutex is a threads synchronization object that can be in two distinct states:

- Not owned (unlocked).
- Owned by a thread (locked).

Operations defined for mutexes:

- **Lock:** The mutex is checked, if the mutex is not owned by some other thread then it is associated to the locking thread else the thread is queued on the mutex in a list ordered by priority.
- **Unlock:** The mutex is released by the owner and the highest priority thread waiting in the queue, if any, is resumed and made owner of the mutex.

Constraints

In ChibiOS/RT the Unlock operations must always be performed in lock-reverse order. This restriction both improves the performance and is required for an efficient implementation of the priority inheritance mechanism.

Operating under this restriction also ensures that deadlocks are no possible.

Recursive mode

By default mutexes are not recursive, this mean that it is not possible to take a mutex already owned by the same thread. It is possible to enable the recursive behavior by enabling the option CH_CFG_USE_MUTEXES_RECURSIVE.

The priority inversion problem

The mutexes in ChibiOS/RT implements the **full** priority inheritance mechanism in order handle the priority inversion problem.

When a thread is queued on a mutex, any thread, directly or indirectly, holding the mutex gains the same priority of the waiting thread (if their priority was not already equal or higher). The mechanism works with any number of nested mutexes and any number of involved threads. The algorithm complexity (worst case) is N with N equal to the number of nested mutexes.

Precondition

In order to use the mutex APIs the CH_CFG_USE_MUTEXES option must be enabled in `chconf.h`.

Postcondition

Enabling mutexes requires 5-12 (depending on the architecture) extra bytes in the `thread_t` structure.

Macros

- `#define _MUTEX_DATA(name) {_THREADS_QUEUE_DATA(name.m_queue), NULL, NULL, 0}`
Data part of a static mutex initializer.
- `#define MUTEX_DECL(name) mutex_t name = _MUTEX_DATA(name)`
Static mutex initializer.

Typedefs

- **typedef struct ch_mutex mutex_t**
Type of a mutex structure.

Data Structures

- **struct ch_mutex**
Mutex structure.

Functions

- **void chMtxObjectInit (mutex_t *mp)**
Initializes a mutex_t structure.
- **void chMtxLock (mutex_t *mp)**
Locks the specified mutex.
- **void chMtxLockS (mutex_t *mp)**
Locks the specified mutex.
- **bool chMtxTryLock (mutex_t *mp)**
Tries to lock a mutex.
- **bool chMtxTryLockS (mutex_t *mp)**
Tries to lock a mutex.
- **void chMtxUnlock (mutex_t *mp)**
Unlocks the specified mutex.
- **void chMtxUnlockS (mutex_t *mp)**
Unlocks the specified mutex.
- **void chMtxUnlockAll (void)**
Unlocks all mutexes owned by the invoking thread.
- **static bool chMtxQueueNotEmptyS (mutex_t *mp)**
Returns true if the mutex queue contains at least a waiting thread.
- **static mutex_t * chMtxGetNextMutexS (void)**
Returns the next mutex in the mutexes stack of the current thread.

9.13.2 Macro Definition Documentation

9.13.2.1 `#define _MUTEX_DATA(name) { _THREADS_QUEUE_DATA(name.m_queue), NULL, NULL, 0 }`

Data part of a static mutex initializer.

This macro should be used when statically initializing a mutex that is part of a bigger structure.

Parameters

in	name	the name of the mutex variable
----	------	--------------------------------

9.13.2.2 `#define MUTEX_DECL(name) mutex_t name = _MUTEX_DATA(name)`

Static mutex initializer.

Statically initialized mutexes require no explicit initialization using `chMtxInit()`.

Parameters

in	<i>name</i>	the name of the mutex variable
----	-------------	--------------------------------

9.13.3 TYPEDOC Documentation**9.13.3.1 `typedef struct ch_mutex mutex_t`**

Type of a mutex structure.

9.13.4 FUNCTION Documentation**9.13.4.1 `void chMtxObjectInit(mutex_t * mp)`**

Initializes a `mutex_t` structure.

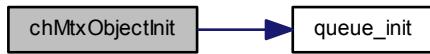
Parameters

out	<i>mp</i>	pointer to a <code>mutex_t</code> structure
-----	-----------	---

Function Class:

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

Here is the call graph for this function:

**9.13.4.2 `void chMtxLock(mutex_t * mp)`**

Locks the specified mutex.

Postcondition

The mutex is locked and inserted in the per-thread stack of owned mutexes.

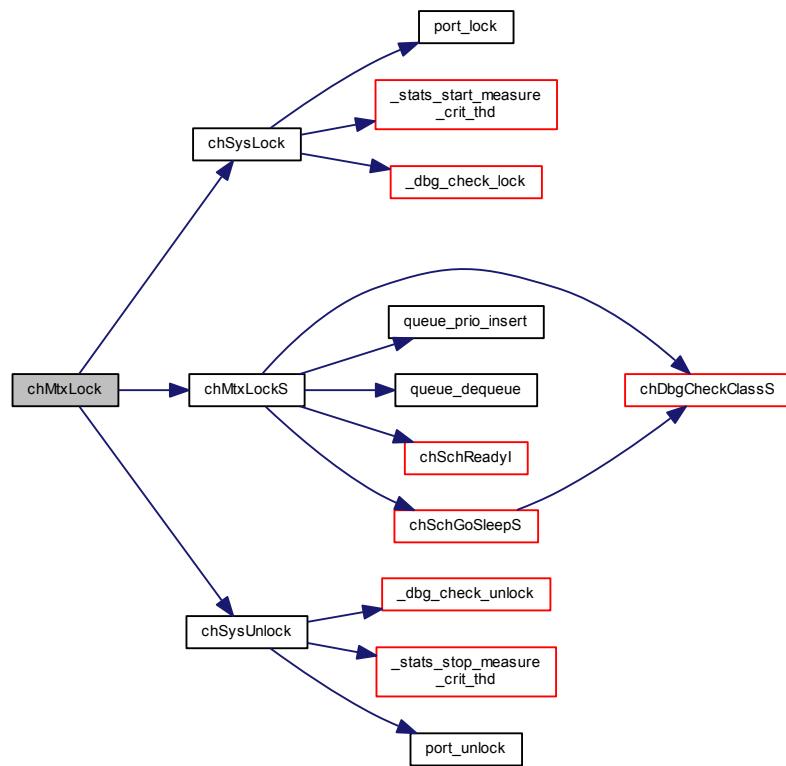
Parameters

in	<i>mp</i>	pointer to the <code>mutex_t</code> structure
----	-----------	---

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



9.13.4.3 void chMtxLockS (mutex_t * mp)

Locks the specified mutex.

Postcondition

The mutex is locked and inserted in the per-thread stack of owned mutexes.

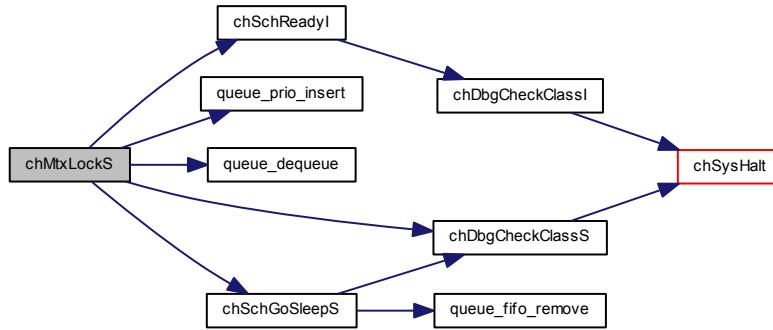
Parameters

in	<i>mp</i>	pointer to the <code>mutex_t</code> structure
----	-----------	---

Function Class:

This is an **S-Class** API, this function can be invoked from within a system lock zone by threads only.

Here is the call graph for this function:



9.13.4.4 bool chMtxTryLock (mutex_t * mp)

Tries to lock a mutex.

This function attempts to lock a mutex, if the mutex is already locked by another thread then the function exits without waiting.

Postcondition

The mutex is locked and inserted in the per-thread stack of owned mutexes.

Note

This function does not have any overhead related to the priority inheritance mechanism because it does not try to enter a sleep state.

Parameters

in	<i>mp</i>	pointer to the <code>mutex_t</code> structure
----	-----------	---

Returns

The operation status.

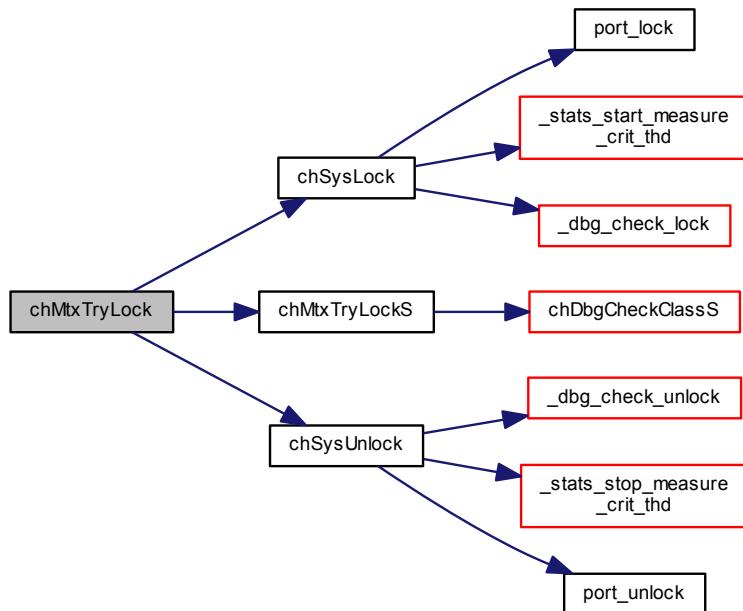
Return values

<i>true</i>	if the mutex has been successfully acquired
<i>false</i>	if the lock attempt failed.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



9.13.4.5 bool chMtxTryLockS (mutex_t * mp)

Tries to lock a mutex.

This function attempts to lock a mutex, if the mutex is already taken by another thread then the function exits without waiting.

Postcondition

The mutex is locked and inserted in the per-thread stack of owned mutexes.

Note

This function does not have any overhead related to the priority inheritance mechanism because it does not try to enter a sleep state.

Parameters

in	<code>mp</code>	pointer to the <code>mutex_t</code> structure
----	-----------------	---

Returns

The operation status.

Return values

<i>true</i>	if the mutex has been successfully acquired
<i>false</i>	if the lock attempt failed.

Function Class:

This is an **S-Class** API, this function can be invoked from within a system lock zone by threads only.

Here is the call graph for this function:

**9.13.4.6 void chMtxUnlock (mutex_t * mp)**

Unlocks the specified mutex.

Note

Mutexes must be unlocked in reverse lock order. Violating this rules will result in a panic if assertions are enabled.

Precondition

The invoking thread **must** have at least one owned mutex.

Postcondition

The mutex is unlocked and removed from the per-thread stack of owned mutexes.

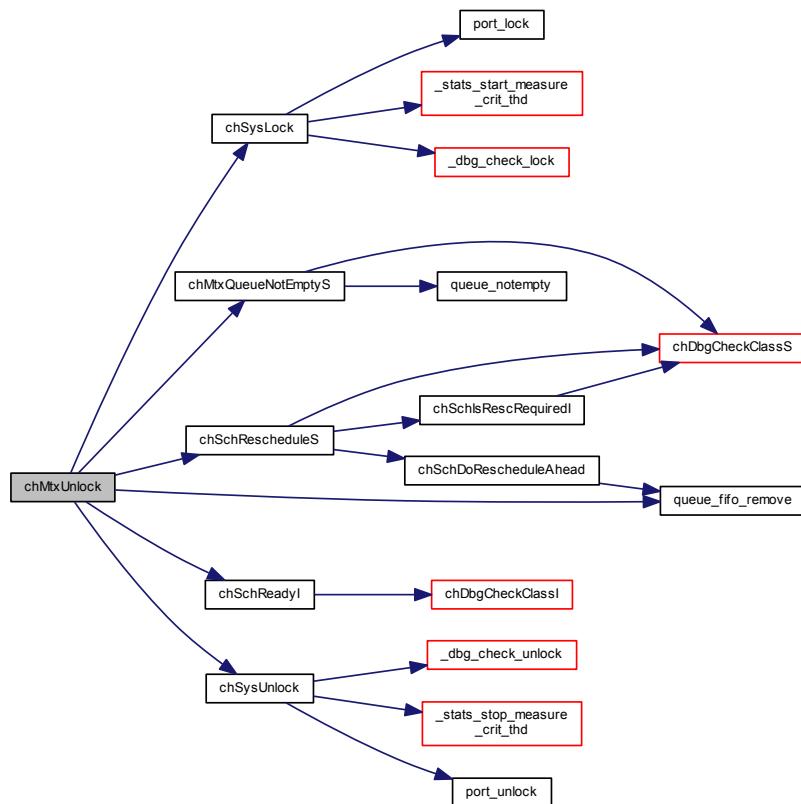
Parameters

in	<i>mp</i>	pointer to the <code>mutex_t</code> structure
----	-----------	---

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

**9.13.4.7 void chMtxUnlockS (mutex_t * mp)**

Unlocks the specified mutex.

Note

Mutexes must be unlocked in reverse lock order. Violating this rules will result in a panic if assertions are enabled.

Precondition

The invoking thread **must** have at least one owned mutex.

Postcondition

The mutex is unlocked and removed from the per-thread stack of owned mutexes.
 This function does not reschedule so a call to a rescheduling function must be performed before unlocking the kernel.

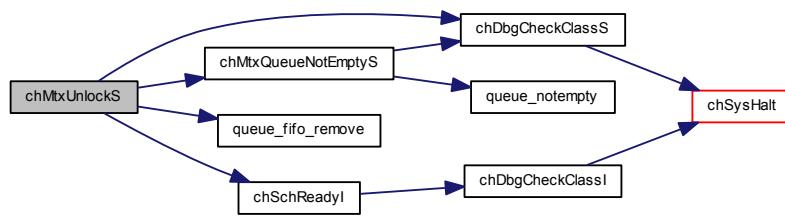
Parameters

in	<i>mp</i>	pointer to the <code>mutex_t</code> structure
----	-----------	---

Function Class:

This is an **S-Class** API, this function can be invoked from within a system lock zone by threads only.

Here is the call graph for this function:

**9.13.4.8 void chMtxUnlockAll(void)**

Unlocks all mutexes owned by the invoking thread.

Postcondition

The stack of owned mutexes is emptied and all the found mutexes are unlocked.

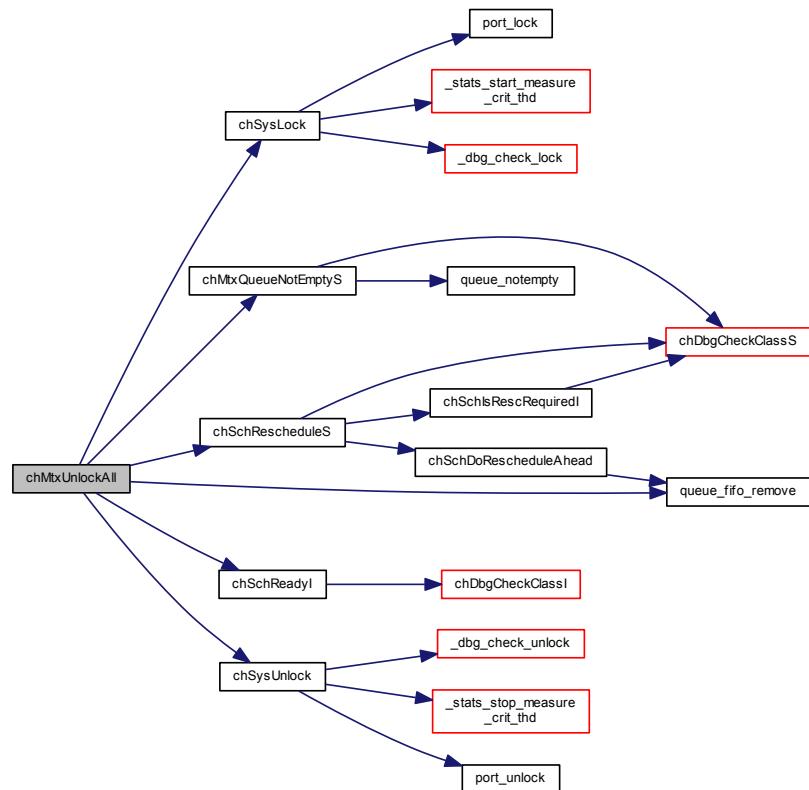
Note

This function is **MUCH MORE** efficient than releasing the mutexes one by one and not just because the call overhead, this function does not have any overhead related to the priority inheritance mechanism.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



9.13.4.9 static bool chMtxQueueNotEmptyS(mutex_t * mp) [inline], [static]

Returns `true` if the mutex queue contains at least a waiting thread.

Parameters

<code>out</code>	<code>mp</code>	pointer to a <code>mutex_t</code> structure
------------------	-----------------	---

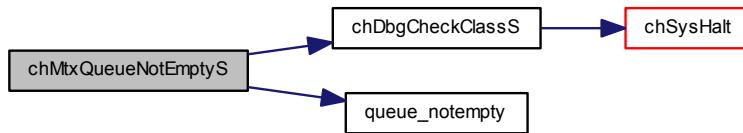
Returns

The mutex queue status.

Function Class:

Deprecated This is an **S-Class** API, this function can be invoked from within a system lock zone by threads only.

Here is the call graph for this function:



9.13.4.10 static mutex_t* chMtxGetNextMutexS (void) [inline], [static]

Returns the next mutex in the mutexes stack of the current thread.

Returns

A pointer to the next mutex in the stack.

Return values

<code>NULL</code>	if the stack is empty.
-------------------	------------------------

Function Class:

This is an **S-Class** API, this function can be invoked from within a system lock zone by threads only.

Here is the call graph for this function:



9.14 Condition Variables

9.14.1 Detailed Description

This module implements the Condition Variables mechanism. Condition variables are an extensions to the mutex subsystem and cannot work alone.

Operation mode

The condition variable is a synchronization object meant to be used inside a zone protected by a mutex. Mutexes and condition variables together can implement a Monitor construct.

Precondition

In order to use the condition variable APIs the CH_CFG_USE_CONDVAR option must be enabled in `chconf.h`.

Macros

- `#define _CONDVAR_DATA(name) {_THREADS_QUEUE_DATA(name.c_queue)}`
Data part of a static condition variable initializer.
- `#define CONDVAR_DECL(name) condition_variable_t name = _CONDVAR_DATA(name)`
Static condition variable initializer.

Typedefs

- `typedef struct condition_variable condition_variable_t`
condition_variable_t structure.

Data Structures

- `struct condition_variable`
condition_variable_t structure.

Functions

- `void chCondObjectInit (condition_variable_t *cp)`
Initializes a condition_variable_t structure.
- `void chCondSignal (condition_variable_t *cp)`
Signals one thread that is waiting on the condition variable.
- `void chCondSignall (condition_variable_t *cp)`
Signals one thread that is waiting on the condition variable.
- `void chCondBroadcast (condition_variable_t *cp)`
Signals all threads that are waiting on the condition variable.
- `void chCondBroadcastl (condition_variable_t *cp)`
Signals all threads that are waiting on the condition variable.
- `msg_t chCondWait (condition_variable_t *cp)`
Waits on the condition variable releasing the mutex lock.
- `msg_t chCondWaitS (condition_variable_t *cp)`
Waits on the condition variable releasing the mutex lock.
- `msg_t chCondWaitTimeout (condition_variable_t *cp, systime_t time)`

Waits on the condition variable releasing the mutex lock.

- `msg_t chCondWaitTimeoutS (condition_variable_t *cp, systime_t time)`

Waits on the condition variable releasing the mutex lock.

9.14.2 Macro Definition Documentation

9.14.2.1 `#define _CONDVAR_DATA(name) {THREADS_QUEUE_DATA(name.c_queue)}`

Data part of a static condition variable initializer.

This macro should be used when statically initializing a condition variable that is part of a bigger structure.

Parameters

in	name	the name of the condition variable
----	------	------------------------------------

9.14.2.2 `#define CONDVAR_DECL(name) condition_variable_t name = _CONDVAR_DATA(name)`

Static condition variable initializer.

Statically initialized condition variables require no explicit initialization using `chCondInit()`.

Parameters

in	name	the name of the condition variable
----	------	------------------------------------

9.14.3 Typedef Documentation

9.14.3.1 `typedef struct condition_variable condition_variable_t`

`condition_variable_t` structure.

9.14.4 Function Documentation

9.14.4.1 `void chCondObjectInit (condition_variable_t * cp)`

Initializes a `condition_variable_t` structure.

Parameters

out	cp	pointer to a <code>condition_variable_t</code> structure
-----	----	--

Function Class:

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

Here is the call graph for this function:



9.14.4.2 void chCondSignal (condition_variable_t * cp)

Signals one thread that is waiting on the condition variable.

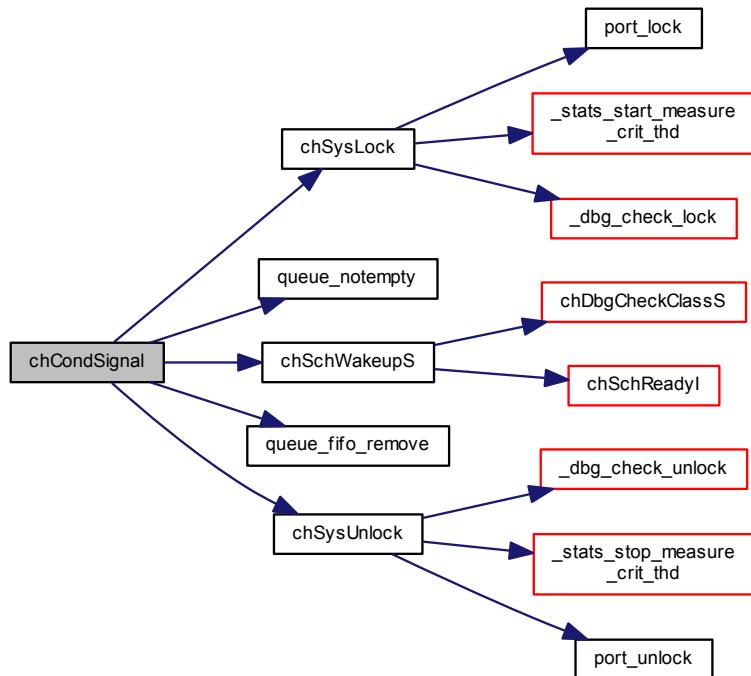
Parameters

in	<i>cp</i>	pointer to the condition_variable_t structure
----	-----------	---

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



9.14.4.3 void chCondSignall (condition_variable_t * cp)

Signals one thread that is waiting on the condition variable.

Postcondition

This function does not reschedule so a call to a rescheduling function must be performed before unlocking the kernel. Note that interrupt handlers always reschedule on exit so an explicit reschedule must not be performed in ISRs.

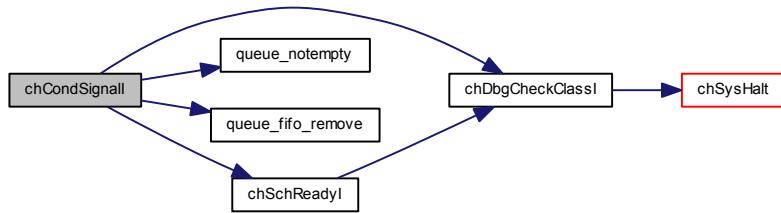
Parameters

in	<i>cp</i>	pointer to the condition_variable_t structure
----	-----------	---

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:

**9.14.4.4 void chCondBroadcast(condition_variable_t * cp)**

Signals all threads that are waiting on the condition variable.

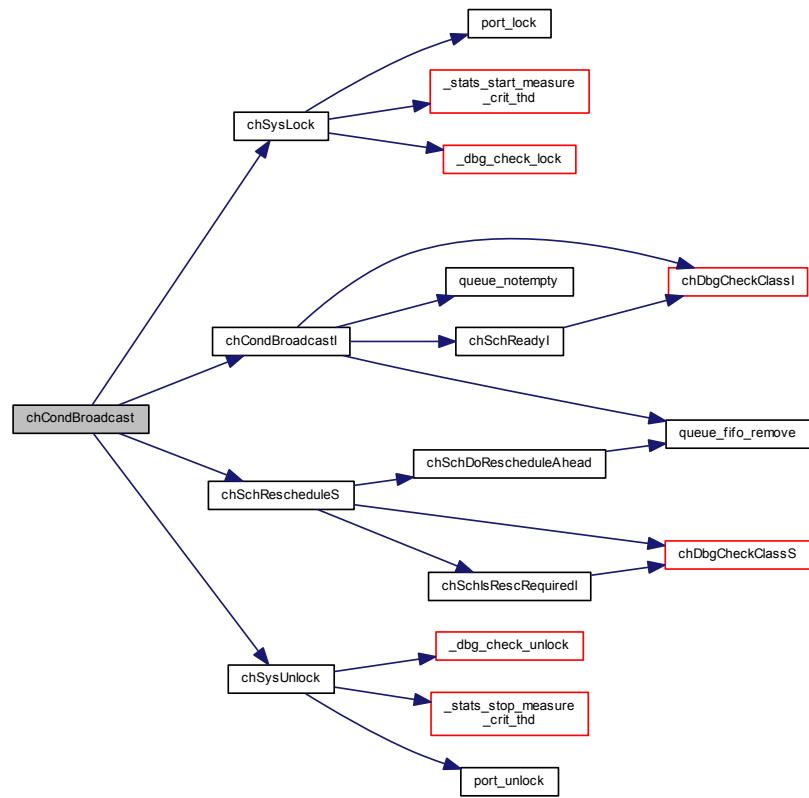
Parameters

in	<i>cp</i>	pointer to the condition_variable_t structure
----	-----------	---

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



9.14.4.5 void chCondBroadcastI (condition_variable_t * cp)

Signals all threads that are waiting on the condition variable.

Postcondition

This function does not reschedule so a call to a rescheduling function must be performed before unlocking the kernel. Note that interrupt handlers always reschedule on exit so an explicit reschedule must not be performed in ISRs.

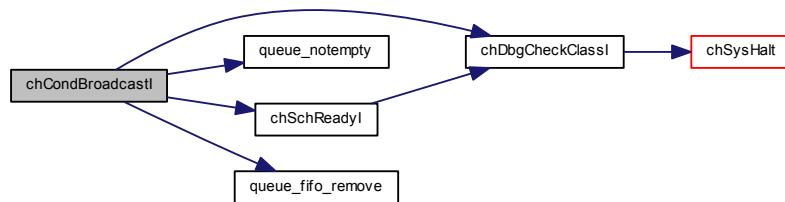
Parameters

in	<i>cp</i>	pointer to the <code>condition_variable_t</code> structure
----	-----------	--

Function Class:

This is an **I-Class API**, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:



9.14.4.6 `msg_t chCondWait(condition_variable_t * cp)`

Waits on the condition variable releasing the mutex lock.

Releases the currently owned mutex, waits on the condition variable, and finally acquires the mutex again. All the sequence is performed atomically.

Precondition

The invoking thread **must** have at least one owned mutex.

Parameters

in	<i>cp</i>	pointer to the <code>condition_variable_t</code> structure
----	-----------	--

Returns

A message specifying how the invoking thread has been released from the condition variable.

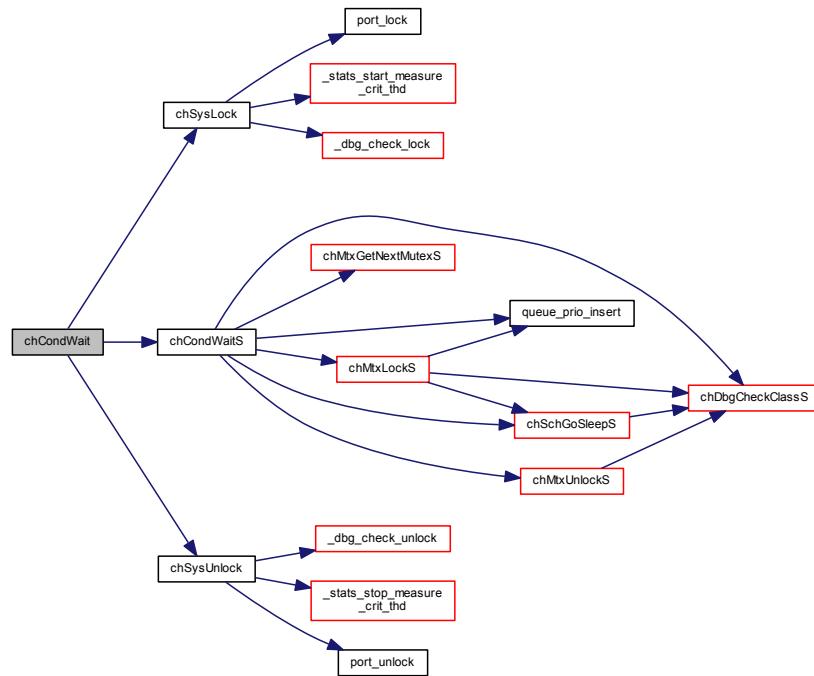
Return values

<code>MSG_OK</code>	if the condition variable has been signaled using <code>chCondSignal()</code> .
<code>MSG_RESET</code>	if the condition variable has been signaled using <code>chCondBroadcast()</code> .

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



9.14.4.7 msg_t chCondWaitS(condition_variable_t * cp)

Waits on the condition variable releasing the mutex lock.

Releases the currently owned mutex, waits on the condition variable, and finally acquires the mutex again. All the sequence is performed atomically.

Precondition

The invoking thread **must** have at least one owned mutex.

Parameters

in	<code>cp</code>	pointer to the <code>condition_variable_t</code> structure
----	-----------------	--

Returns

A message specifying how the invoking thread has been released from the condition variable.

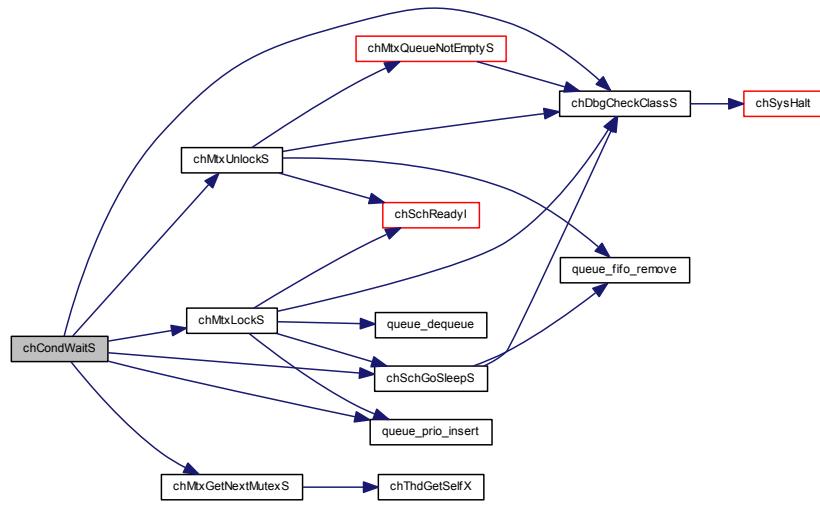
Return values

<code>MSG_OK</code>	if the condition variable has been signaled using <code>chCondSignal()</code> .
<code>MSG_RESET</code>	if the condition variable has been signaled using <code>chCondBroadcast()</code> .

Function Class:

This is an **S-Class** API, this function can be invoked from within a system lock zone by threads only.

Here is the call graph for this function:



9.14.4.8 msg_t chCondWaitTimeout (condition_variable_t * cp, systime_t time)

Waits on the condition variable releasing the mutex lock.

Releases the currently owned mutex, waits on the condition variable, and finally acquires the mutex again. All the sequence is performed atomically.

Precondition

The invoking thread **must** have at least one owned mutex.

The configuration option `CH_CFG_USE_CONDVAR_TIMEOUT` must be enabled in order to use this function.

Postcondition

Exiting the function because a timeout does not re-acquire the mutex, the mutex ownership is lost.

Parameters

in	<i>cp</i>	pointer to the <code>condition_variable_t</code> structure
in	<i>time</i>	the number of ticks before the operation timeouts, the special values are handled as follow: <ul style="list-style-type: none"> • <code>TIME_INFINITE</code> no timeout. • <code>TIME_IMMEDIATE</code> this value is not allowed.

Returns

A message specifying how the invoking thread has been released from the condition variable.

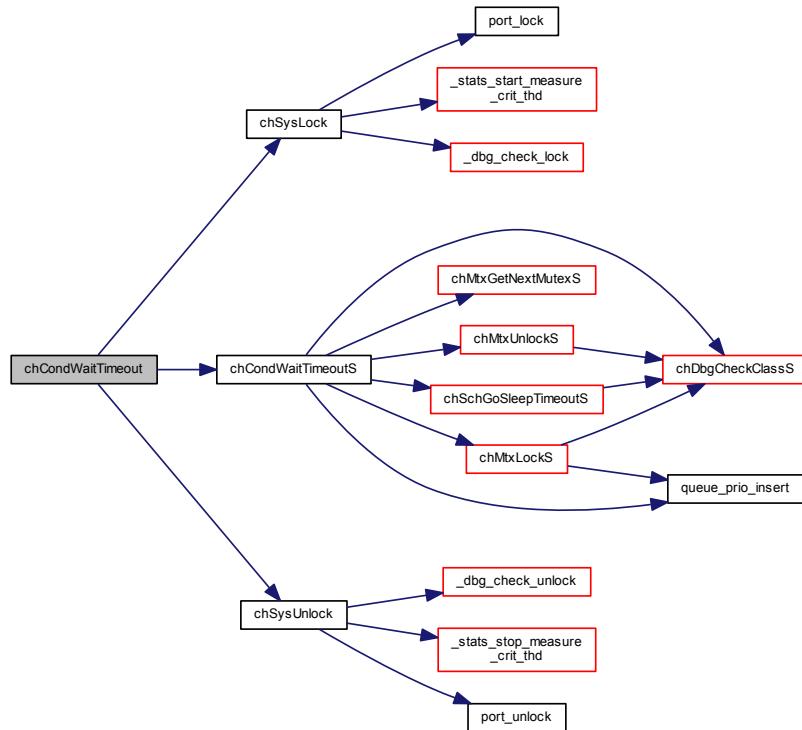
Return values

<code>MSG_OK</code>	if the condition variable has been signaled using <code>chCondSignal()</code> .
<code>MSG_RESET</code>	if the condition variable has been signaled using <code>chCondBroadcast()</code> .
<code>MSG_TIMEOUT</code>	if the condition variable has not been signaled within the specified timeout.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



9.14.4.9 `msg_t chCondWaitTimeoutS(condition_variable_t * cp, systime_t time)`

Waits on the condition variable releasing the mutex lock.

Releases the currently owned mutex, waits on the condition variable, and finally acquires the mutex again. All the sequence is performed atomically.

Precondition

The invoking thread **must** have at least one owned mutex.

The configuration option `CH_CFG_USE_CONDVAR_TIMEOUT` must be enabled in order to use this function.

Postcondition

Exiting the function because a timeout does not re-acquire the mutex, the mutex ownership is lost.

Parameters

in	<i>cp</i>	pointer to the <code>condition_variable_t</code> structure
in	<i>time</i>	<p>the number of ticks before the operation timeouts, the special values are handled as follow:</p> <ul style="list-style-type: none"> • <code>TIME_INFINITE</code> no timeout. • <code>TIME_IMMEDIATE</code> this value is not allowed.

Returns

A message specifying how the invoking thread has been released from the condition variable.

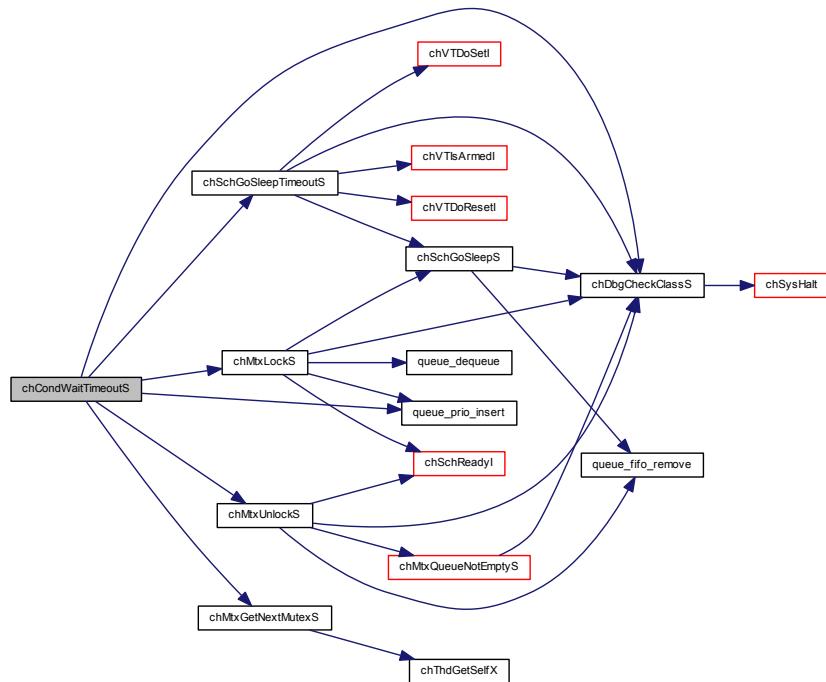
Return values

<code>MSG_OK</code>	if the condition variable has been signaled using <code>chCondSignal()</code> .
<code>MSG_RESET</code>	if the condition variable has been signaled using <code>chCondBroadcast()</code> .
<code>MSG_TIMEOUT</code>	if the condition variable has not been signaled within the specified timeout.

Function Class:

This is an **S-Class API**, this function can be invoked from within a system lock zone by threads only.

Here is the call graph for this function:



9.15 Event Flags

9.15.1 Detailed Description

Event Flags, Event Sources and Event Listeners.

Operation mode

Each thread has a mask of pending events inside its `thread_t` structure. Operations defined for events:

- **Wait**, the invoking thread goes to sleep until a certain AND/OR combination of events become pending.
- **Clear**, a mask of events is cleared from the pending events, the cleared events mask is returned (only the events that were actually pending and then cleared).
- **Signal**, an events mask is directly ORed to the mask of the signaled thread.
- **Broadcast**, each thread registered on an Event Source is signaled with the events specified in its Event Listener.
- **Dispatch**, an events mask is scanned and for each bit set to one an associated handler function is invoked. Bit masks are scanned from bit zero upward.

An Event Source is a special object that can be "broadcasted" by a thread or an interrupt service routine. Broadcasting an Event Source has the effect that all the threads registered on the Event Source will be signaled with an events mask.

An unlimited number of Event Sources can exists in a system and each thread can be listening on an unlimited number of them.

Precondition

In order to use the Events APIs the `CH_CFG_USE_EVENTS` option must be enabled in `chconf.h`.

Postcondition

Enabling events requires 1-4 (depending on the architecture) extra bytes in the `thread_t` structure.

Macros

- `#define ALL_EVENTS ((eventmask_t)-1)`
All events allowed mask.
- `#define EVENT_MASK(eid) ((eventmask_t)1 << (eventmask_t)(eid))`
Returns an event mask from an event identifier.
- `#define _EVENTSOURCE_DATA(name) {(void *)(&name)}`
Data part of a static event source initializer.
- `#define EVENTSOURCE_DECL(name) event_source_t name = _EVENTSOURCE_DATA(name)`
Static event source initializer.

Typedefs

- `typedef struct event_source event_source_t`
Event Source structure.
- `typedef void(* evhandler_t) (eventid_t id)`
Event Handler callback function.

Data Structures

- struct `event_listener`
Event Listener structure.
- struct `event_source`
Event Source structure.

Functions

- void `chEvtRegisterMaskWithFlags` (`event_source_t` *esp, `event_listener_t` *elp, `eventmask_t` events, `eventflags_t` wflags)
Registers an Event Listener on an Event Source.
- void `chEvtUnregister` (`event_source_t` *esp, `event_listener_t` *elp)
Unregisters an Event Listener from its Event Source.
- `eventmask_t chEvtGetAndClearEvents` (`eventmask_t` events)
Clears the pending events specified in the events mask.
- `eventmask_t chEvtAddEvents` (`eventmask_t` events)
*Adds (OR) a set of events to the current thread, this is **much** faster than using `chEvtBroadcast()` or `chEvtSignal()`.*
- void `chEvtBroadcastFlagsI` (`event_source_t` *esp, `eventflags_t` flags)
Signals all the Event Listeners registered on the specified Event Source.
- `eventflags_t chEvtGetAndClearFlags` (`event_listener_t` *elp)
Returns the flags associated to an `event_listener_t`.
- void `chEvtSignal` (`thread_t` *tp, `eventmask_t` events)
Adds a set of event flags directly to the specified `thread_t`.
- void `chEvtSignall` (`thread_t` *tp, `eventmask_t` events)
Adds a set of event flags directly to the specified `thread_t`.
- void `chEvtBroadcastFlags` (`event_source_t` *esp, `eventflags_t` flags)
Signals all the Event Listeners registered on the specified Event Source.
- `eventflags_t chEvtGetAndClearFlagsI` (`event_listener_t` *elp)
Returns the flags associated to an `event_listener_t`.
- void `chEvtDispatch` (`const evhandler_t` *handlers, `eventmask_t` events)
Invokes the event handlers associated to an event flags mask.
- `eventmask_t chEvtWaitOne` (`eventmask_t` events)
Waits for exactly one of the specified events.
- `eventmask_t chEvtWaitAny` (`eventmask_t` events)
Waits for any of the specified events.
- `eventmask_t chEvtWaitAll` (`eventmask_t` events)
Waits for all the specified events.
- `eventmask_t chEvtWaitOneTimeout` (`eventmask_t` events, `systime_t` time)
Waits for exactly one of the specified events.
- `eventmask_t chEvtWaitAnyTimeout` (`eventmask_t` events, `systime_t` time)
Waits for any of the specified events.
- `eventmask_t chEvtWaitAllTimeout` (`eventmask_t` events, `systime_t` time)
Waits for all the specified events.
- static void `chEvtObjectInit` (`event_source_t` *esp)
Initializes an Event Source.
- static void `chEvtRegisterMask` (`event_source_t` *esp, `event_listener_t` *elp, `eventmask_t` events)
Registers an Event Listener on an Event Source.
- static void `chEvtRegister` (`event_source_t` *esp, `event_listener_t` *elp, `eventid_t` event)
Registers an Event Listener on an Event Source.

- static bool `chEvtIsListeningl` (`event_source_t` *`esp`)
Verifies if there is at least one `event_listener_t` registered.
- static void `chEvtBroadcast` (`event_source_t` *`esp`)
Signals all the Event Listeners registered on the specified Event Source.
- static void `chEvtBroadcastl` (`event_source_t` *`esp`)
Signals all the Event Listeners registered on the specified Event Source.

9.15.2 Macro Definition Documentation

9.15.2.1 `#define ALL_EVENTS ((eventmask_t)-1)`

All events allowed mask.

9.15.2.2 `#define EVENT_MASK(eid) ((eventmask_t)1 << (eventmask_t)(eid))`

Returns an event mask from an event identifier.

9.15.2.3 `#define _EVENTSOURCE_DATA(name) {(void *)(&name)}`

Data part of a static event source initializer.

This macro should be used when statically initializing an event source that is part of a bigger structure.

Parameters

<code>name</code>	the name of the event source variable
-------------------	---------------------------------------

9.15.2.4 `#define EVENTSOURCE_DECL(name) event_source_t name = _EVENTSOURCE_DATA(name)`

Static event source initializer.

Statically initialized event sources require no explicit initialization using `chEvtInit()`.

Parameters

<code>name</code>	the name of the event source variable
-------------------	---------------------------------------

9.15.3 Typedef Documentation

9.15.3.1 `typedef struct event_source event_source_t`

Event Source structure.

9.15.3.2 `typedef void(* evhandler_t)(eventid_t id)`

Event Handler callback function.

9.15.4 Function Documentation

9.15.4.1 `void chEvtRegisterMaskWithFlags(event_source_t * esp, event_listener_t * elp, eventmask_t events, eventflags_t wflags)`

Registers an Event Listener on an Event Source.

Once a thread has registered as listener on an event source it will be notified of all events broadcasted there.

Note

Multiple Event Listeners can specify the same bits to be ORed to different threads.

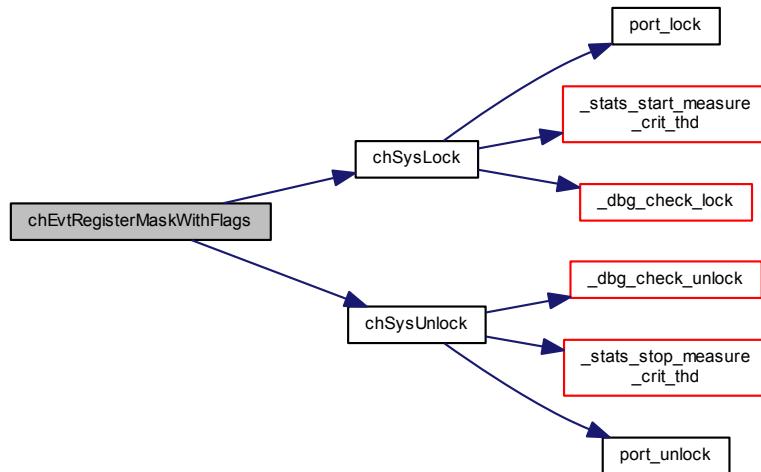
Parameters

in	<i>esp</i>	pointer to the <code>event_source_t</code> structure
in	<i>elp</i>	pointer to the <code>event_listener_t</code> structure
in	<i>events</i>	events to be ORed to the thread when the event source is broadcasted
in	<i>wflags</i>	mask of flags the listening thread is interested in

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



9.15.4.2 void chEvtUnregister(`event_source_t` * *esp*, `event_listener_t` * *elp*)

Unregisters an Event Listener from its Event Source.

Note

If the event listener is not registered on the specified event source then the function does nothing.
For optimal performance it is better to perform the unregister operations in inverse order of the register operations (elements are found on top of the list).

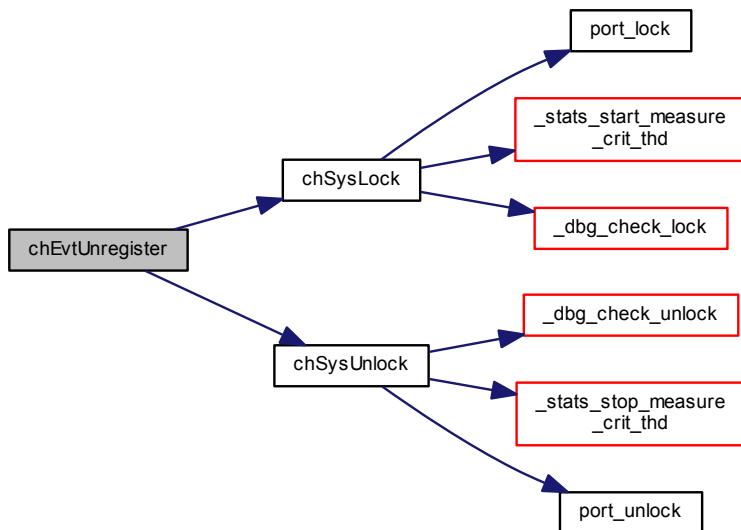
Parameters

in	<i>esp</i>	pointer to the <code>event_source_t</code> structure
in	<i>e/p</i>	pointer to the <code>event_listener_t</code> structure

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

**9.15.4.3 eventmask_t chEvtGetAndClearEvents (`eventmask_t events`)**

Clears the pending events specified in the events mask.

Parameters

in	<i>events</i>	the events to be cleared
----	---------------	--------------------------

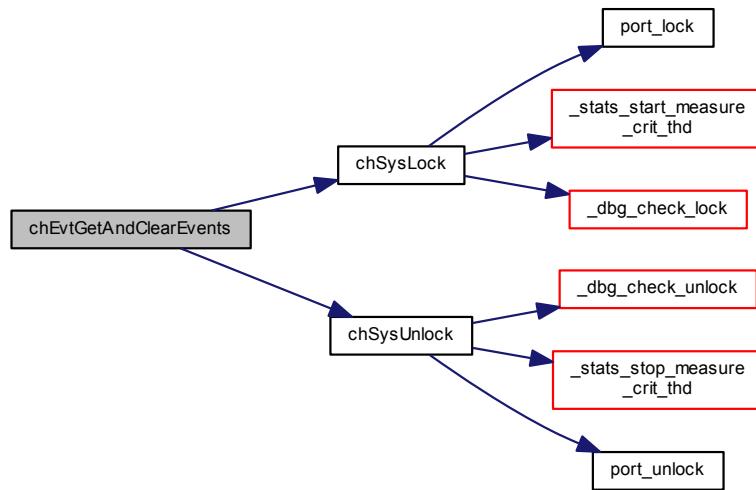
Returns

The pending events that were cleared.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

**9.15.4.4 eventmask_t chEvtAddEvents (eventmask_t events)**

Adds (OR) a set of events to the current thread, this is **much** faster than using `chEvtBroadcast()` or `chEvtSignal()`.

Parameters

in	<i>events</i>	the events to be added
----	---------------	------------------------

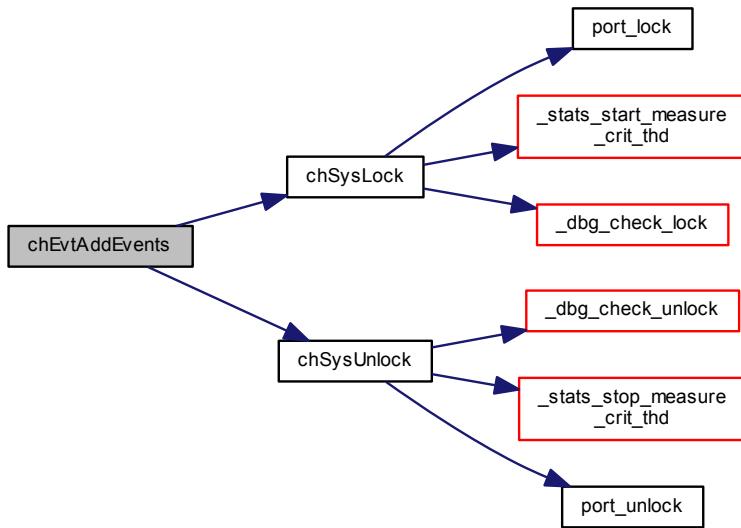
Returns

The current pending events.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



9.15.4.5 void chEvtBroadcastFlags(event_source_t * esp, eventflags_t flags)

Signals all the Event Listeners registered on the specified Event Source.

This function variants ORs the specified event flags to all the threads registered on the `event_source_t` in addition to the event flags specified by the threads themselves in the `event_listener_t` objects.

Postcondition

This function does not reschedule so a call to a rescheduling function must be performed before unlocking the kernel. Note that interrupt handlers always reschedule on exit so an explicit reschedule must not be performed in ISRs.

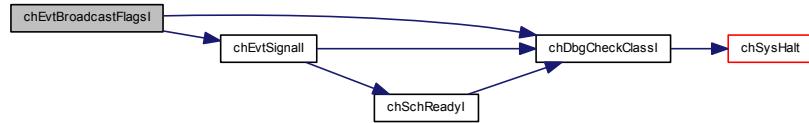
Parameters

in	<code>esp</code>	pointer to the <code>event_source_t</code> structure
in	<code>flags</code>	the flags set to be added to the listener flags mask

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:



9.15.4.6 eventflags_t chEvtGetAndClearFlags (event_listener_t * elp)

Returns the flags associated to an `event_listener_t`.

The flags are returned and the `event_listener_t` flags mask is cleared.

Parameters

in	<code>elp</code>	pointer to the <code>event_listener_t</code> structure
----	------------------	--

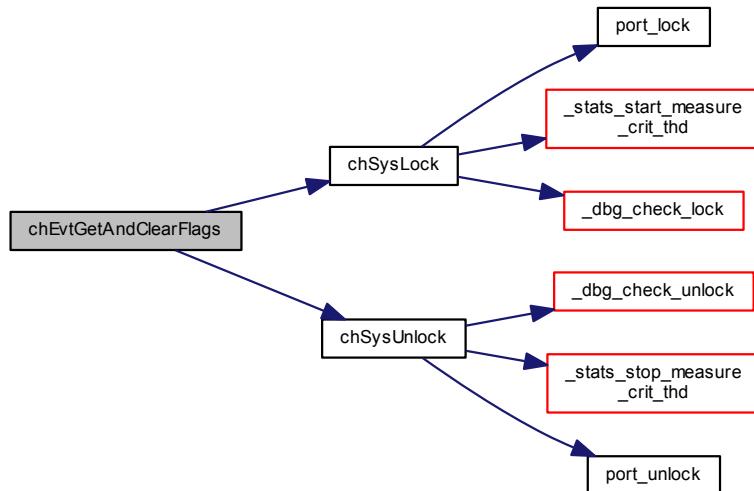
Returns

The flags added to the listener by the associated event source.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



9.15.4.7 void chEvtSignal (thread_t * tp, eventmask_t events)

Adds a set of event flags directly to the specified `thread_t`.

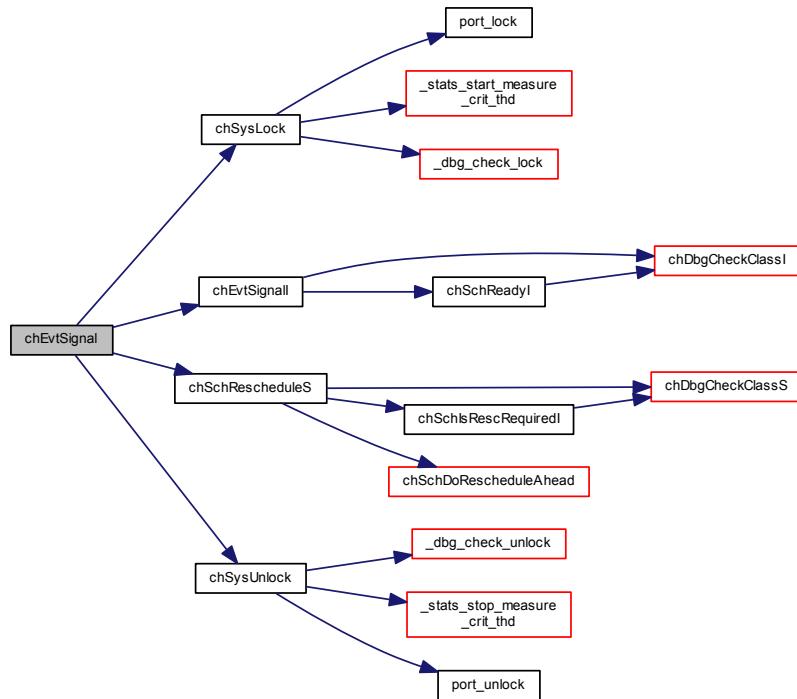
Parameters

in	<i>tp</i>	the thread to be signaled
in	<i>events</i>	the events set to be ORed

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



9.15.4.8 void chEvtSignall (*thread_t* * *tp*, *eventmask_t* *events*)

Adds a set of event flags directly to the specified *thread_t*.

Postcondition

This function does not reschedule so a call to a rescheduling function must be performed before unlocking the kernel. Note that interrupt handlers always reschedule on exit so an explicit reschedule must not be performed in ISRs.

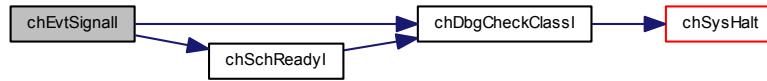
Parameters

in	<i>tp</i>	the thread to be signaled
in	<i>events</i>	the events set to be ORed

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:



9.15.4.9 void chEvtBroadcastFlags (event_source_t * esp, eventflags_t flags)

Signals all the Event Listeners registered on the specified Event Source.

This function variants ORs the specified event flags to all the threads registered on the `event_source_t` in addition to the event flags specified by the threads themselves in the `event_listener_t` objects.

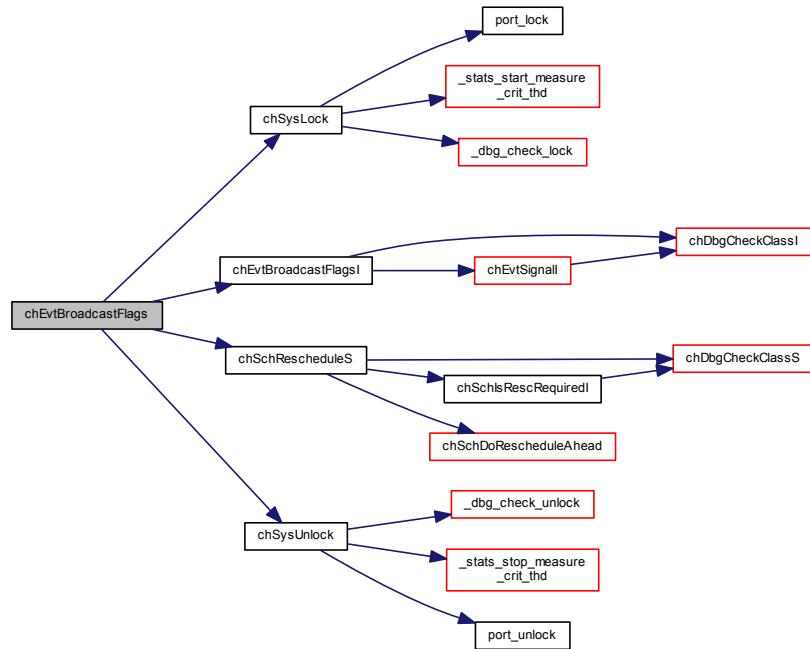
Parameters

in	<code>esp</code>	pointer to the <code>event_source_t</code> structure
in	<code>flags</code>	the flags set to be added to the listener flags mask

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



9.15.4.10 `eventflags_t chEvtGetAndClearFlags(event_listener_t * elp)`

Returns the flags associated to an `event_listener_t`.

The flags are returned and the `event_listener_t` flags mask is cleared.

Parameters

in	<code>elp</code>	pointer to the <code>event_listener_t</code> structure
----	------------------	--

Returns

The flags added to the listener by the associated event source.

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

9.15.4.11 `void chEvtDispatch(const evhandler_t * handlers, eventmask_t events)`

Invokes the event handlers associated to an event flags mask.

Parameters

in	<code>events</code>	mask of events to be dispatched
in	<code>handlers</code>	an array of <code>evhandler_t</code> . The array must have size equal to the number of bits in <code>eventmask_t</code> .

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

9.15.4.12 `eventmask_t chEvtWaitOne(eventmask_t events)`

Waits for exactly one of the specified events.

The function waits for one event among those specified in `events` to become pending then the event is cleared and returned.

Note

One and only one event is served in the function, the one with the lowest event id. The function is meant to be invoked into a loop in order to serve all the pending events.

This means that Event Listeners with a lower event identifier have an higher priority.

Parameters

in	<code>events</code>	events that the function should wait for, <code>ALL_EVENTS</code> enables all the events
----	---------------------	--

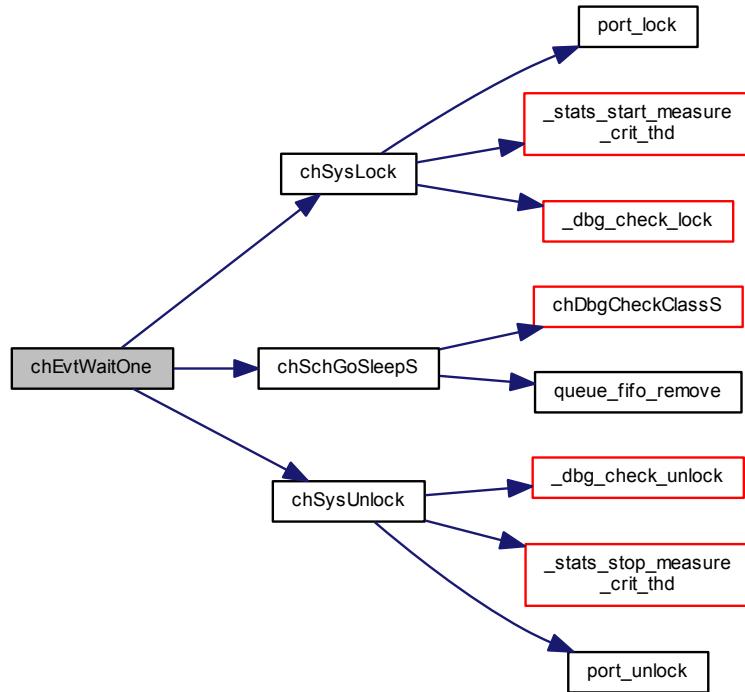
Returns

The mask of the lowest event id served and cleared.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



9.15.4.13 `eventmask_t chEvtWaitAny (eventmask_t events)`

Waits for any of the specified events.

The function waits for any event among those specified in `events` to become pending then the events are cleared and returned.

Parameters

<code>in</code>	<code>events</code>	events that the function should wait for, <code>ALL_EVENTS</code> enables all the events
-----------------	---------------------	--

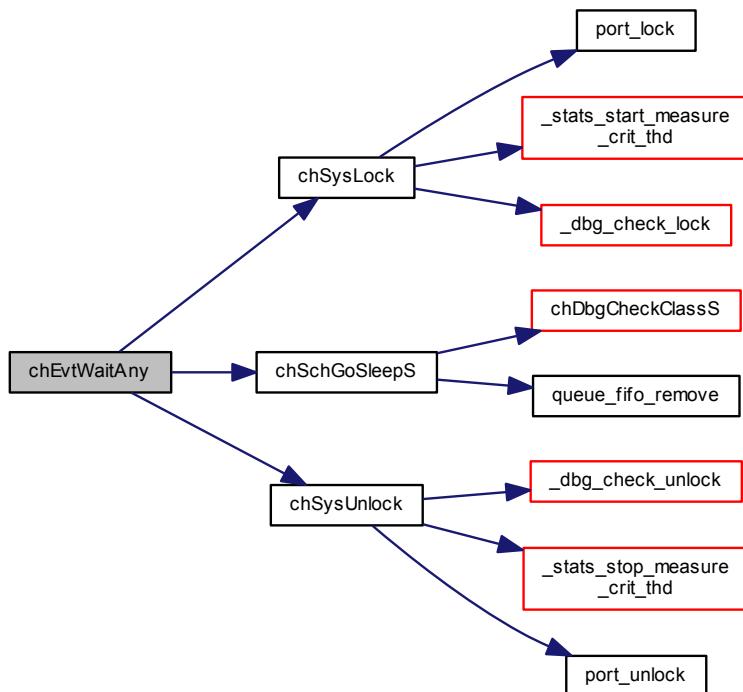
Returns

The mask of the served and cleared events.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

**9.15.4.14 eventmask_t chEvtWaitAll(eventmask_t events)**

Waits for all the specified events.

The function waits for all the events specified in `events` to become pending then the events are cleared and returned.

Parameters

in	<code>events</code>	events that the function should wait for, ALL_EVENTS requires all the events
----	---------------------	--

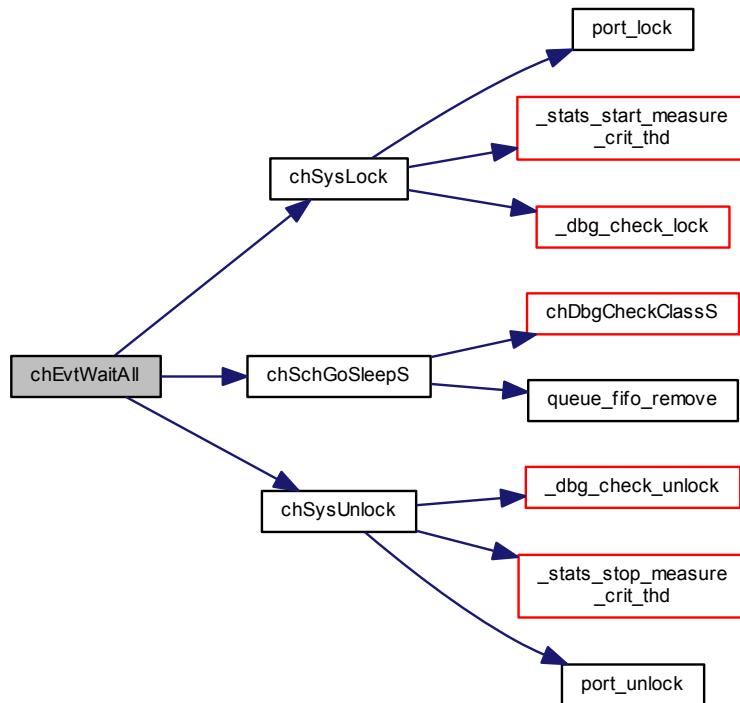
Returns

The mask of the served and cleared events.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



9.15.4.15 eventmask_t chEvtWaitOneTimeout(eventmask_t events, systime_t time)

Waits for exactly one of the specified events.

The function waits for one event among those specified in `events` to become pending then the event is cleared and returned.

Note

One and only one event is served in the function, the one with the lowest event id. The function is meant to be invoked into a loop in order to serve all the pending events.

This means that Event Listeners with a lower event identifier have an higher priority.

Parameters

<code>in</code>	<code>events</code>	events that the function should wait for, <code>ALL_EVENTS</code> enables all the events
<code>in</code>	<code>time</code>	<p>the number of ticks before the operation timeouts, the following special values are allowed:</p> <ul style="list-style-type: none"> • <code>TIME_IMMEDIATE</code> immediate timeout. • <code>TIME_INFINITE</code> no timeout.

Returns

The mask of the lowest event id served and cleared.

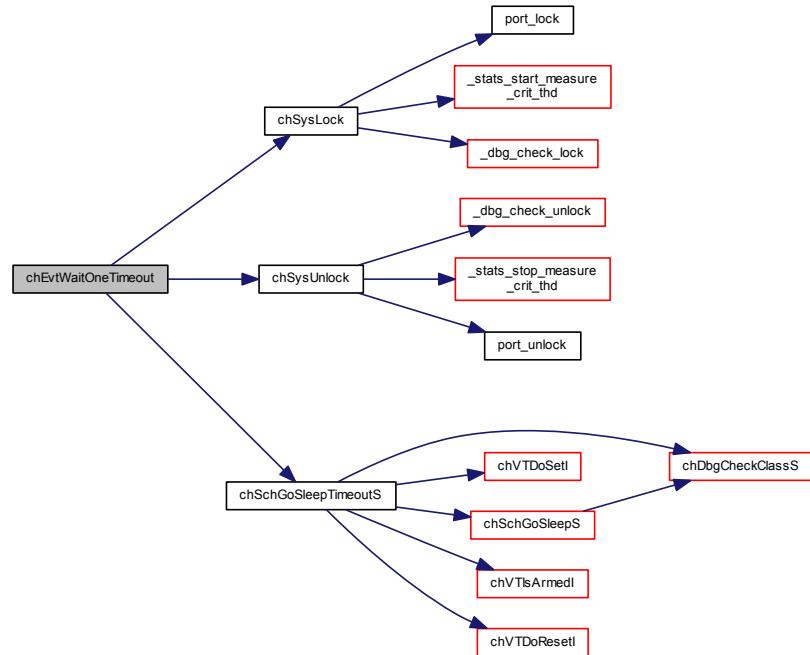
Return values

0	if the operation has timed out.
---	---------------------------------

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



9.15.4.16 eventmask_t chEvtWaitAnyTimeout (eventmask_t events, systime_t time)

Waits for any of the specified events.

The function waits for any event among those specified in `events` to become pending then the events are cleared and returned.

Parameters

in	<code>events</code>	events that the function should wait for, <code>ALL_EVENTS</code> enables all the events
in	<code>time</code>	<p>the number of ticks before the operation timeouts, the following special values are allowed:</p> <ul style="list-style-type: none"> • <code>TIME_IMMEDIATE</code> immediate timeout. • <code>TIME_INFINITE</code> no timeout.

Returns

The mask of the served and cleared events.

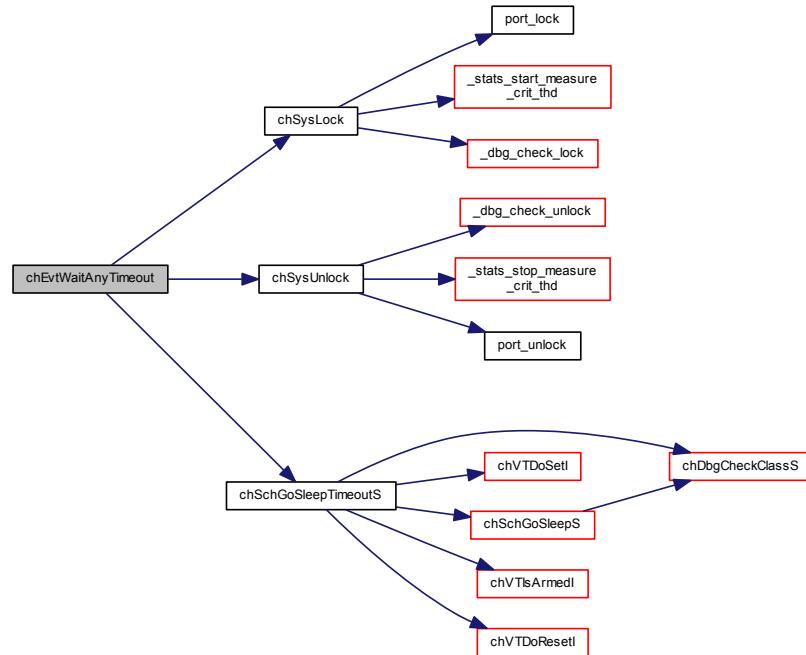
Return values

0	if the operation has timed out.
---	---------------------------------

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



9.15.4.17 eventmask_t chEvtWaitAllTimeout(eventmask_t events, systime_t time)

Waits for all the specified events.

The function waits for all the events specified in `events` to become pending then the events are cleared and returned.

Parameters

in	<code>events</code>	events that the function should wait for, <code>ALL_EVENTS</code> requires all the events
in	<code>time</code>	<p>the number of ticks before the operation timeouts, the following special values are allowed:</p> <ul style="list-style-type: none"> • <code>TIME_IMMEDIATE</code> immediate timeout. • <code>TIME_INFINITE</code> no timeout.

Returns

The mask of the served and cleared events.

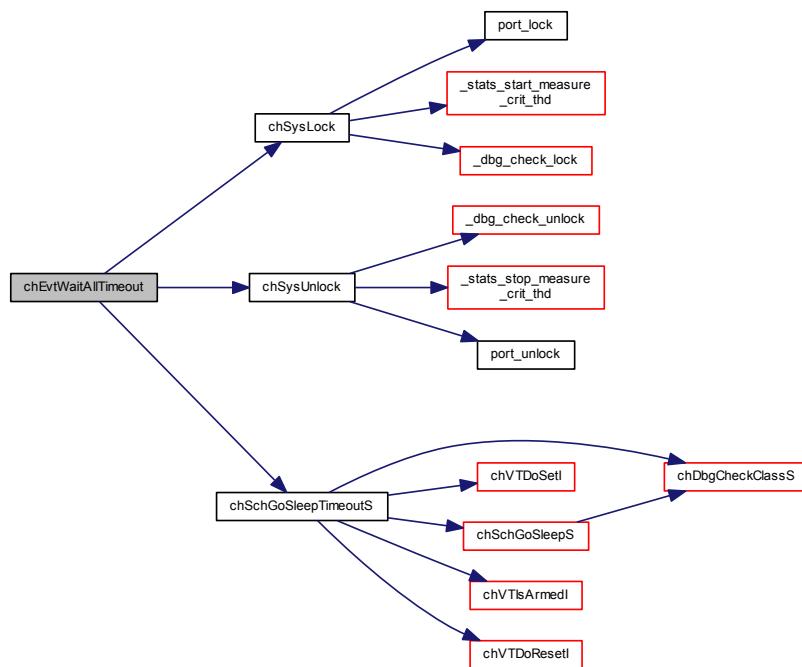
Return values

<i>O</i>	if the operation has timed out.
----------	---------------------------------

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



9.15.4.18 static void chEvtObjectInit(event_source_t * esp) [inline], [static]

Initializes an Event Source.

Note

This function can be invoked before the kernel is initialized because it just prepares a `event_source_t` structure.

Parameters

in	<code>esp</code>	pointer to the <code>event_source_t</code> structure
----	------------------	--

Function Class:

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

9.15.4.19 static void chEvtRegisterMask(event_source_t * esp, event_listener_t * elp, eventmask_t events) [inline], [static]

Registers an Event Listener on an Event Source.

Once a thread has registered as listener on an event source it will be notified of all events broadcasted there.

Note

Multiple Event Listeners can specify the same bits to be ORed to different threads.

Parameters

in	<i>esp</i>	pointer to the <code>event_source_t</code> structure
out	<i>e/p</i>	pointer to the <code>event_listener_t</code> structure
in	<i>events</i>	the mask of events to be ORed to the thread when the event source is broadcasted

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



9.15.4.20 static void chEvtRegister (`event_source_t` * *esp*, `event_listener_t` * *e/p*, `eventid_t` *event*) [inline], [static]

Registers an Event Listener on an Event Source.

Note

Multiple Event Listeners can use the same event identifier, the listener will share the callback function.

Parameters

in	<i>esp</i>	pointer to the <code>event_source_t</code> structure
out	<i>e/p</i>	pointer to the <code>event_listener_t</code> structure
in	<i>event</i>	numeric identifier assigned to the Event Listener. The value must range between zero and the size, in bit, of the <code>eventmask_t</code> type minus one.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



9.15.4.21 static bool chEvtIsListening(event_source_t * esp) [inline], [static]

Verifies if there is at least one event_listener_t registered.

Parameters

in	esp	pointer to the event_source_t structure
----	-----	---

Returns

The event source status.

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

9.15.4.22 static void chEvtBroadcast(event_source_t * esp) [inline], [static]

Signals all the Event Listeners registered on the specified Event Source.

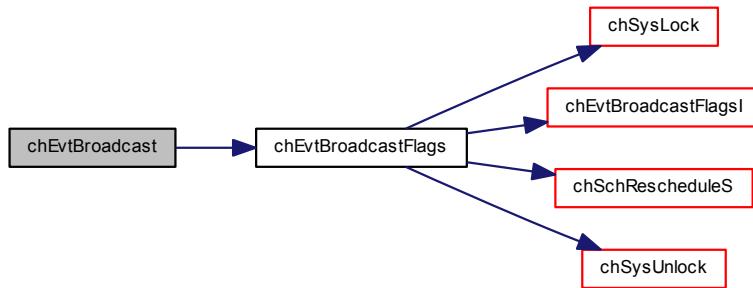
Parameters

in	esp	pointer to the event_source_t structure
----	-----	---

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



9.15.4.23 static void chEvtBroadcastI(event_source_t * esp) [inline], [static]

Signals all the Event Listeners registered on the specified Event Source.

Postcondition

This function does not reschedule so a call to a rescheduling function must be performed before unlocking the kernel. Note that interrupt handlers always reschedule on exit so an explicit reschedule must not be performed in ISRs.

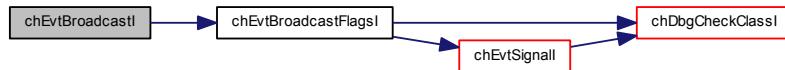
Parameters

in	esp	pointer to the event_source_t structure
----	-----	---

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:



9.16 Synchronous Messages

9.16.1 Detailed Description

Synchronous inter-thread messages APIs and services.

Operation Mode

Synchronous messages are an easy to use and fast IPC mechanism, threads can both act as message servers and/or message clients, the mechanism allows data to be carried in both directions. Note that messages are not copied between the client and server threads but just a pointer passed so the exchange is very time efficient. Messages are scalar data types of type `msg_t` that are guaranteed to be size compatible with data pointers. Note that on some architectures function pointers can be larger than `msg_t`. Messages are usually processed in FIFO order but it is possible to process them in priority order by enabling the `CH_CFG_USE_MESSAGES_PRIORITY` option in [chconf.h](#).

Precondition

In order to use the message APIs the `CH_CFG_USE_MESSAGES` option must be enabled in [chconf.h](#).

Postcondition

Enabling messages requires 6-12 (depending on the architecture) extra bytes in the `thread_t` structure.

Functions

- `msg_t chMsgSend (thread_t *tp, msg_t msg)`
Sends a message to the specified thread.
- `thread_t * chMsgWait (void)`
Suspends the thread and waits for an incoming message.
- `void chMsgRelease (thread_t *tp, msg_t msg)`
Releases a sender thread specifying a response message.
- `static bool chMsgIsPending (thread_t *tp)`
Evaluates to true if the thread has pending messages.
- `static msg_t chMsgGet (thread_t *tp)`
Returns the message carried by the specified thread.
- `static void chMsgReleaseS (thread_t *tp, msg_t msg)`
Releases the thread waiting on top of the messages queue.

9.16.2 Function Documentation

9.16.2.1 `msg_t chMsgSend (thread_t * tp, msg_t msg)`

Sends a message to the specified thread.

The sender is stopped until the receiver executes a `chMsgRelease ()` after receiving the message.

Parameters

in	<code>tp</code>	the pointer to the thread
----	-----------------	---------------------------

in	msg	the message
----	-----	-------------

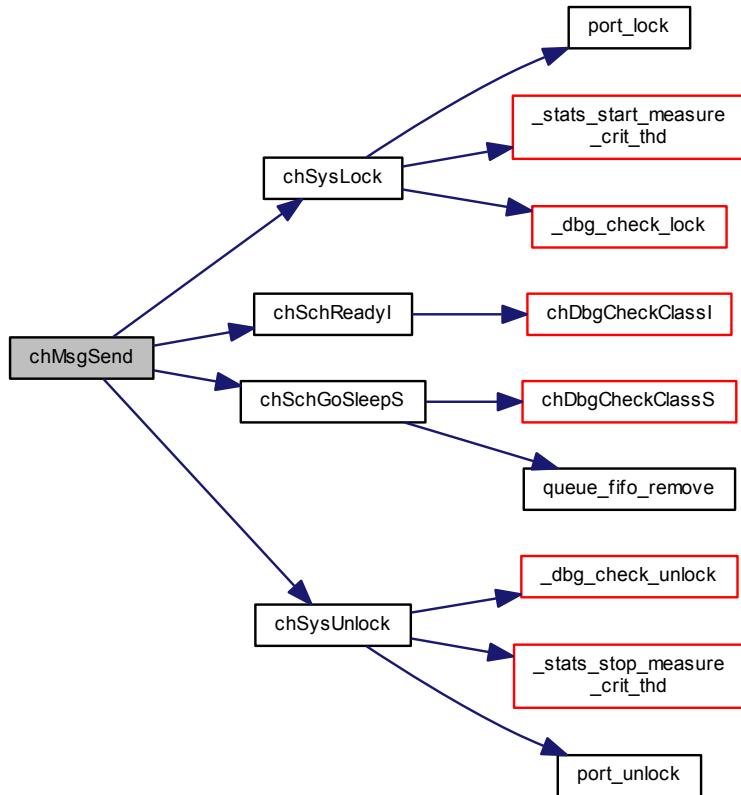
Returns

The answer message from [chMsgRelease\(\)](#).

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

**9.16.2.2 thread_t * chMsgWait(void)**

Suspends the thread and waits for an incoming message.

Postcondition

After receiving a message the function [chMsgGet\(\)](#) must be called in order to retrieve the message and then [chMsgRelease\(\)](#) must be invoked in order to acknowledge the reception and send the answer.

Note

If the message is a pointer then you can assume that the data pointed by the message is stable until you invoke [chMsgRelease\(\)](#) because the sending thread is suspended until then.

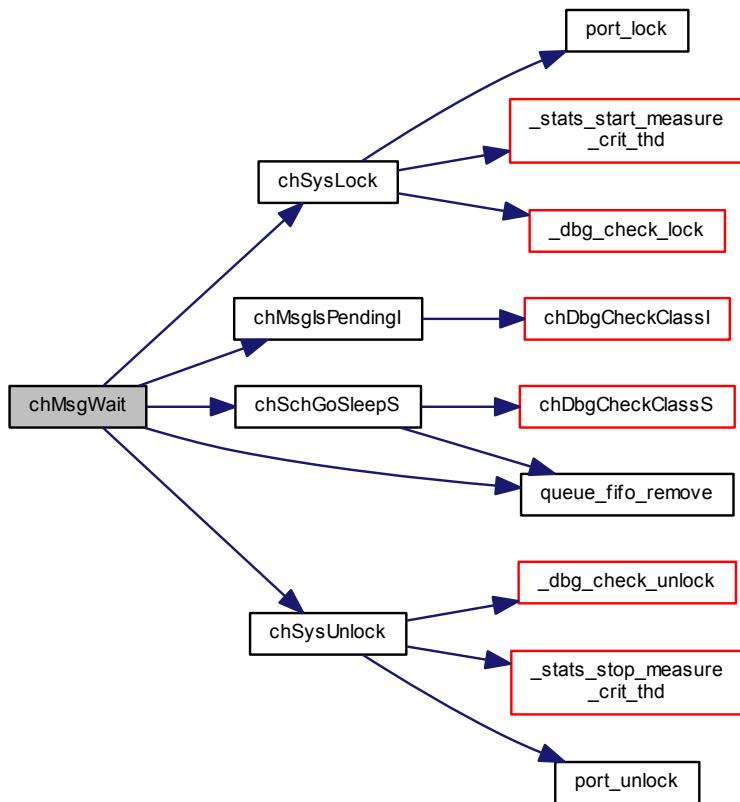
Returns

A reference to the thread carrying the message.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



9.16.2.3 void chMsgRelease (`thread_t * tp, msg_t msg`)

Releases a sender thread specifying a response message.

Precondition

Invoke this function only after a message has been received using [chMsgWait\(\)](#).

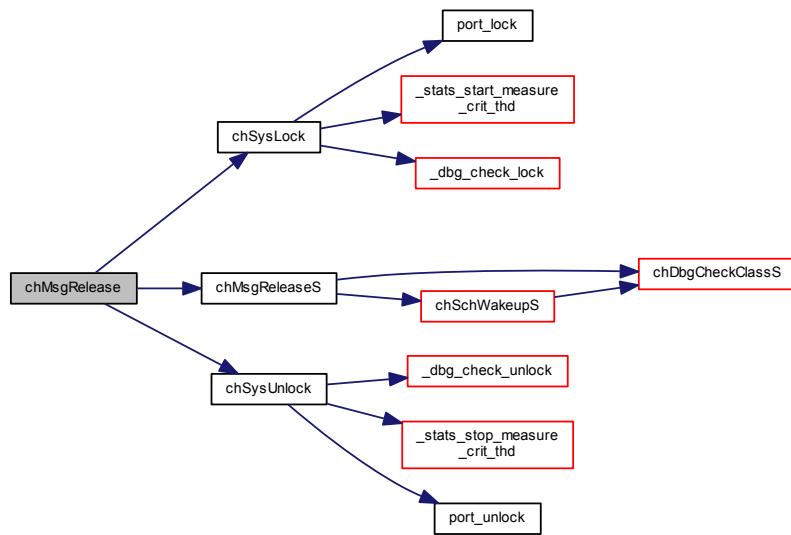
Parameters

in	<i>tp</i>	pointer to the thread
in	<i>msg</i>	message to be returned to the sender

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



9.16.2.4 static bool chMsgIsPendingI(thread_t * tp) [inline], [static]

Evaluates to `true` if the thread has pending messages.

Parameters

in	<i>tp</i>	pointer to the thread
----	-----------	-----------------------

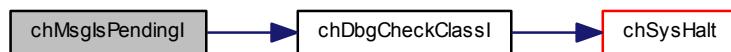
Returns

The pending messages status.

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:

**9.16.2.5 static msg_t chMsgGet(thread_t * tp) [inline], [static]**

Returns the message carried by the specified thread.

Precondition

This function must be invoked immediately after exiting a call to [chMsgWait\(\)](#).

Parameters

in	<i>tp</i>	pointer to the thread
----	-----------	-----------------------

Returns

The message carried by the sender.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

9.16.2.6 static void chMsgReleaseS(thread_t * tp, msg_t msg) [inline], [static]

Releases the thread waiting on top of the messages queue.

Precondition

Invoke this function only after a message has been received using [chMsgWait\(\)](#).

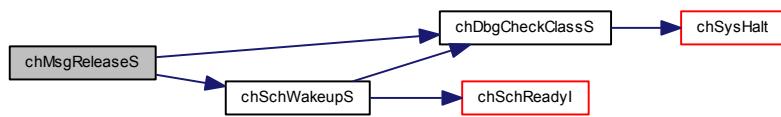
Parameters

in	<i>tp</i>	pointer to the thread
in	<i>msg</i>	message to be returned to the sender

Function Class:

This is an **S-Class** API, this function can be invoked from within a system lock zone by threads only.

Here is the call graph for this function:



9.17 Mailboxes

9.17.1 Detailed Description

Asynchronous messages.

Operation mode

A mailbox is an asynchronous communication mechanism.

Operations defined for mailboxes:

- **Post**: Posts a message on the mailbox in FIFO order.
- **Post Ahead**: Posts a message on the mailbox with urgent priority.
- **Fetch**: A message is fetched from the mailbox and removed from the queue.
- **Reset**: The mailbox is emptied and all the stored messages are lost.

A message is a variable of type `msg_t` that is guaranteed to have the same size of and be compatible with (data) pointers (anyway an explicit cast is needed). If larger messages need to be exchanged then a pointer to a structure can be posted in the mailbox but the posting side has no predefined way to know when the message has been processed. A possible approach is to allocate memory (from a memory pool for example) from the posting side and free it on the fetching side. Another approach is to set a "done" flag into the structure pointed by the message.

Precondition

In order to use the mailboxes APIs the `CH_CFG_USE_MAILBOXES` option must be enabled in `chconf.h`.

Macros

- `#define _MAILBOX_DATA(name, buffer, size)`
Data part of a static mailbox initializer.
- `#define MAILBOX_DECL(name, buffer, size) mailbox_t name = _MAILBOX_DATA(name, buffer, size)`
Static mailbox initializer.

Data Structures

- struct `mailbox_t`
Structure representing a mailbox object.

Functions

- `void chMBOObjectInit (mailbox_t *mbp, msg_t *buf, cnt_t n)`
Initializes a `mailbox_t` object.
- `void chMBReset (mailbox_t *mbp)`
Resets a `mailbox_t` object.
- `void chMBResetI (mailbox_t *mbp)`
Resets a `mailbox_t` object.
- `msg_t chMBPost (mailbox_t *mbp, msg_t msg, systime_t timeout)`
Posts a message into a mailbox.
- `msg_t chMBPostS (mailbox_t *mbp, msg_t msg, systime_t timeout)`
Posts a message into a mailbox.

- `msg_t chMBPostI (mailbox_t *mbp, msg_t msg)`
Posts a message into a mailbox.
- `msg_t chMBPostAhead (mailbox_t *mbp, msg_t msg, systime_t timeout)`
Posts an high priority message into a mailbox.
- `msg_t chMBPostAheadS (mailbox_t *mbp, msg_t msg, systime_t timeout)`
Posts an high priority message into a mailbox.
- `msg_t chMBPostAheadI (mailbox_t *mbp, msg_t msg)`
Posts an high priority message into a mailbox.
- `msg_t chMBFetch (mailbox_t *mbp, msg_t *msgp, systime_t timeout)`
Retrieves a message from a mailbox.
- `msg_t chMBFetchS (mailbox_t *mbp, msg_t *msgp, systime_t timeout)`
Retrieves a message from a mailbox.
- `msg_t chMBFetchI (mailbox_t *mbp, msg_t *msgp)`
Retrieves a message from a mailbox.
- `static size_t chMBGetSizel (mailbox_t *mbp)`
Returns the mailbox buffer size.
- `static cnt_t chMBGetFreeCountl (mailbox_t *mbp)`
Returns the number of free message slots into a mailbox.
- `static cnt_t chMBGetUsedCountl (mailbox_t *mbp)`
Returns the number of used message slots into a mailbox.
- `static msg_t chMBPeekI (mailbox_t *mbp)`
Returns the next message in the queue without removing it.

9.17.2 Macro Definition Documentation

9.17.2.1 #define _MAILBOX_DATA(name, buffer, size)

Value:

```
{
  (msg_t *) (buffer), \
  (msg_t *) (buffer) + size, \
  (msg_t *) (buffer), \
  (msg_t *) (buffer), \
  _SEMAPHORE_DATA(name.mb_fullsem, 0), \
  _SEMAPHORE_DATA(name.mb_emptysem, size), \
}
```

Data part of a static mailbox initializer.

This macro should be used when statically initializing a mailbox that is part of a bigger structure.

Parameters

in	name	the name of the mailbox variable
in	buffer	pointer to the mailbox buffer area
in	size	size of the mailbox buffer area

9.17.2.2 #define MAILBOX_DECL(name, buffer, size) mailbox_t name = _MAILBOX_DATA(name, buffer, size)

Static mailbox initializer.

Statically initialized mailboxes require no explicit initialization using `chMBInit()`.

Parameters

in	<i>name</i>	the name of the mailbox variable
in	<i>buffer</i>	pointer to the mailbox buffer area
in	<i>size</i>	size of the mailbox buffer area

9.17.3 Function Documentation**9.17.3.1 void chMBOBJECTInit (*mailbox_t* * *mbp*, *msg_t* * *buf*, *cnt_t* *n*)**

Initializes a [mailbox_t](#) object.

Parameters

out	<i>mbp</i>	the pointer to the mailbox_t structure to be initialized
in	<i>buf</i>	pointer to the messages buffer as an array of msg_t
in	<i>n</i>	number of elements in the buffer array

Function Class:

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

Here is the call graph for this function:

**9.17.3.2 void chMBReset (*mailbox_t* * *mbp*)**

Resets a [mailbox_t](#) object.

All the waiting threads are resumed with status `MSG_RESET` and the queued messages are lost.

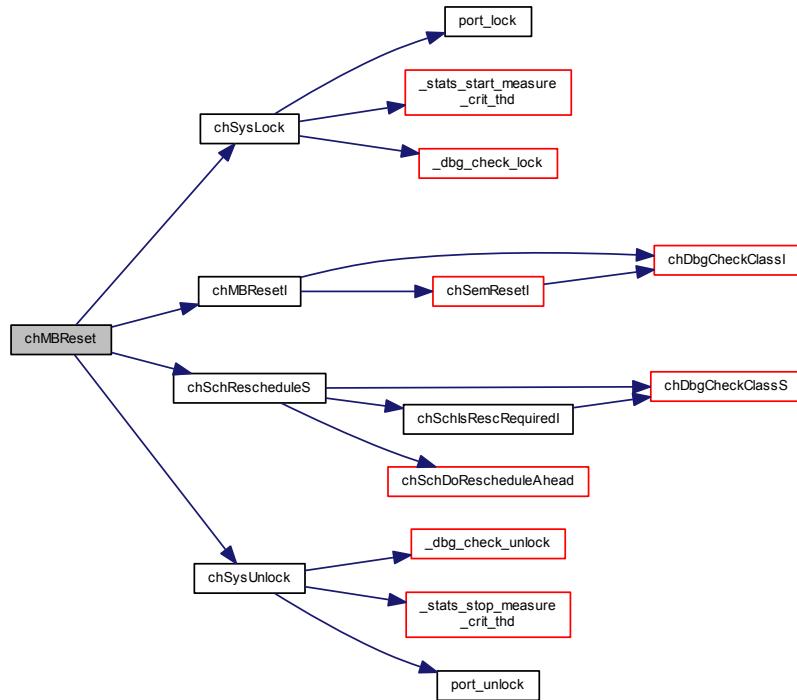
Parameters

in	<i>mbp</i>	the pointer to an initialized mailbox_t object
----	------------	--

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



9.17.3.3 void chMBResetl(mailbox_t * mbp)

Resets a `mailbox_t` object.

All the waiting threads are resumed with status `MSG_RESET` and the queued messages are lost.

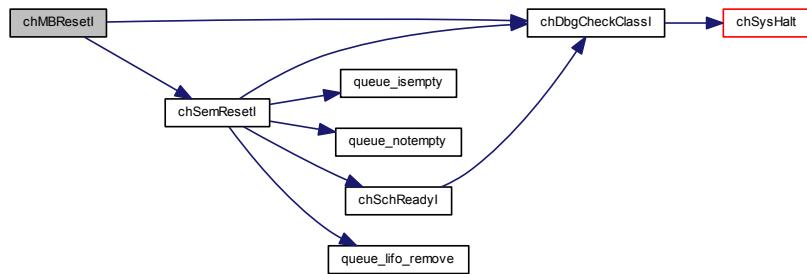
Parameters

in	<code>mbp</code>	the pointer to an initialized <code>mailbox_t</code> object
----	------------------	---

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



9.17.3.4 `msg_t chMBPost(mailbox_t *mbp, msg_t msg, systime_t timeout)`

Posts a message into a mailbox.

The invoking thread waits until a empty slot in the mailbox becomes available or the specified time runs out.

Parameters

in	<code>mbp</code>	the pointer to an initialized <code>mailbox_t</code> object
in	<code>msg</code>	the message to be posted on the mailbox
in	<code>timeout</code>	the number of ticks before the operation timeouts, the following special values are allowed: <ul style="list-style-type: none"> • <code>TIME_IMMEDIATE</code> immediate timeout. • <code>TIME_INFINITE</code> no timeout.

Returns

The operation status.

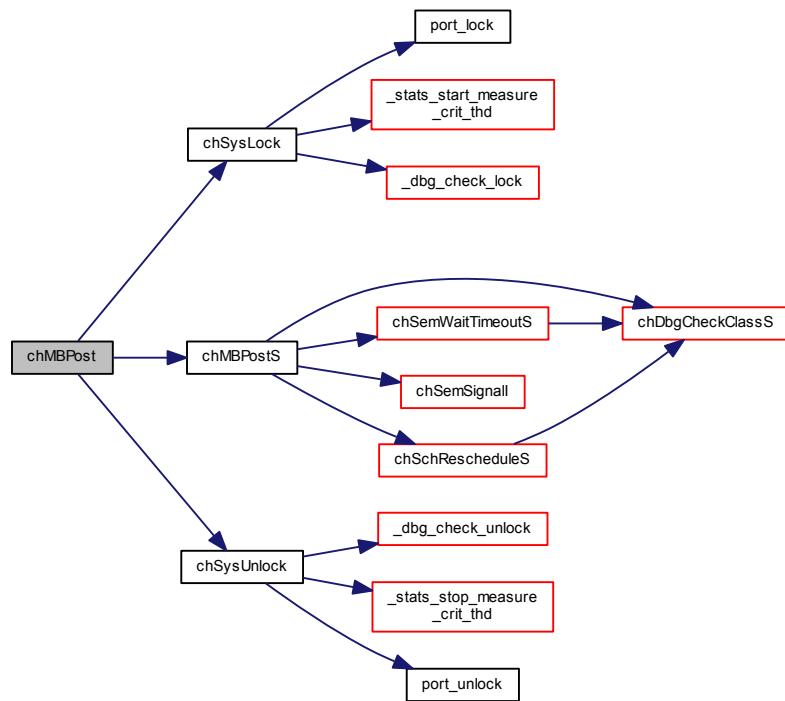
Return values

<code>MSG_OK</code>	if a message has been correctly posted.
<code>MSG_RESET</code>	if the mailbox has been reset while waiting.
<code>MSG_TIMEOUT</code>	if the operation has timed out.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



9.17.3.5 msg_t chMBPostS(mailbox_t * mbp, msg_t msg, systime_t timeout)

Posts a message into a mailbox.

The invoking thread waits until a empty slot in the mailbox becomes available or the specified time runs out.

Parameters

in	<i>mbp</i>	the pointer to an initialized mailbox_t object
in	<i>msg</i>	the message to be posted on the mailbox
in	<i>timeout</i>	the number of ticks before the operation timeouts, the following special values are allowed: <ul style="list-style-type: none"> • <i>TIME_IMMEDIATE</i> immediate timeout. • <i>TIME_INFINITE</i> no timeout.

Returns

The operation status.

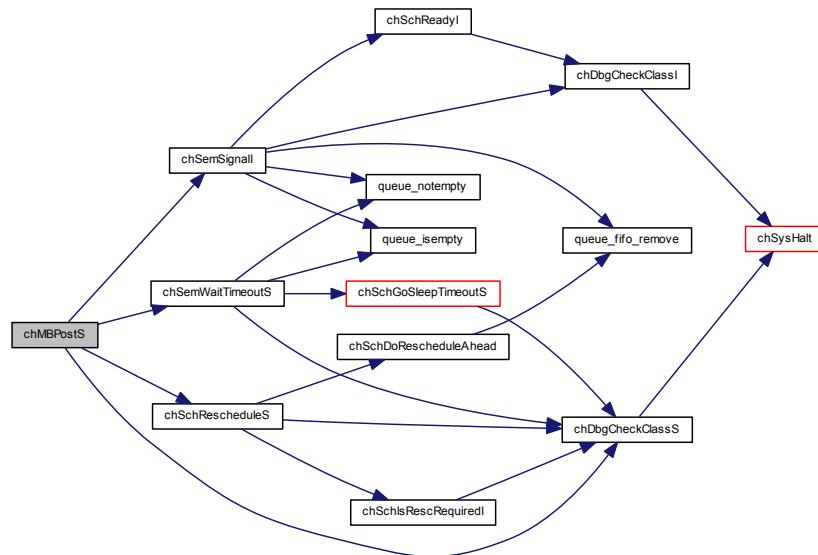
Return values

<i>MSG_OK</i>	if a message has been correctly posted.
<i>MSG_RESET</i>	if the mailbox has been reset while waiting.
<i>MSG_TIMEOUT</i>	if the operation has timed out.

Function Class:

This is an **S-Class** API, this function can be invoked from within a system lock zone by threads only.

Here is the call graph for this function:



9.17.3.6 `msg_t chMBPostI(mailbox_t *mbp, msg_t msg)`

Posts a message into a mailbox.

This variant is non-blocking, the function returns a timeout condition if the queue is full.

Parameters

in	<code>mbp</code>	the pointer to an initialized <code>mailbox_t</code> object
in	<code>msg</code>	the message to be posted on the mailbox

Returns

The operation status.

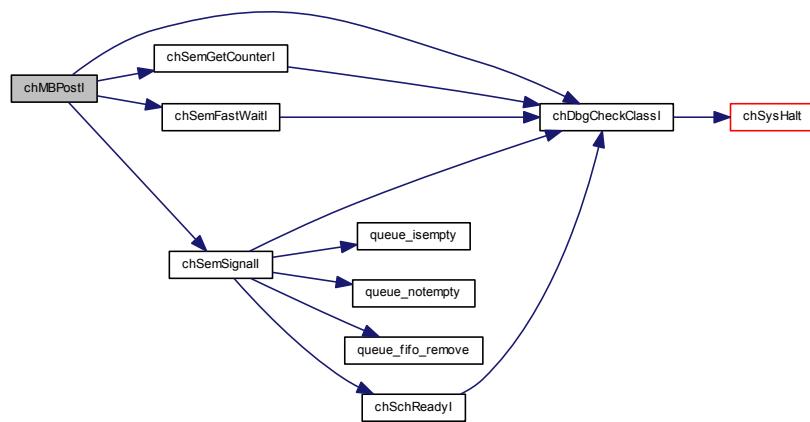
Return values

<code>MSG_OK</code>	if a message has been correctly posted.
<code>MSG_TIMEOUT</code>	if the mailbox is full and the message cannot be posted.

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:



9.17.3.7 msg_t chMBPostAhead (mailbox_t * mbp, msg_t msg, systime_t timeout)

Posts an high priority message into a mailbox.

The invoking thread waits until a empty slot in the mailbox becomes available or the specified time runs out.

Parameters

in	<i>mbp</i>	the pointer to an initialized <code>mailbox_t</code> object
in	<i>msg</i>	the message to be posted on the mailbox
in	<i>timeout</i>	the number of ticks before the operation timeouts, the following special values are allowed: <ul style="list-style-type: none"> • <code>TIME_IMMEDIATE</code> immediate timeout. • <code>TIME_INFINITE</code> no timeout.

Returns

The operation status.

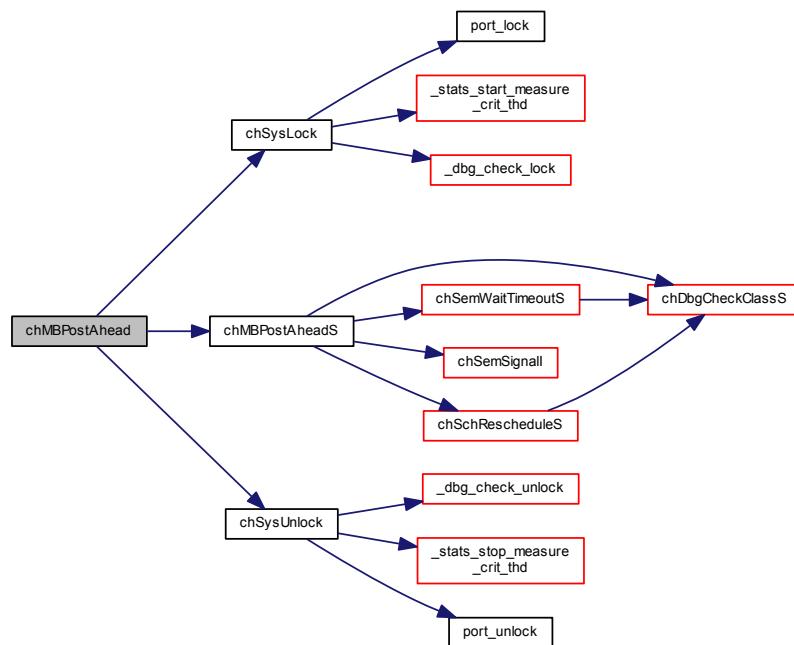
Return values

<i>MSG_OK</i>	if a message has been correctly posted.
<i>MSG_RESET</i>	if the mailbox has been reset while waiting.
<i>MSG_TIMEOUT</i>	if the operation has timed out.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

**9.17.3.8 msg_t chMBPostAheadS (mailbox_t * mbp, msg_t msg, systime_t timeout)**

Posts an high priority message into a mailbox.

The invoking thread waits until a empty slot in the mailbox becomes available or the specified time runs out.

Parameters

in	<i>mbp</i>	the pointer to an initialized <code>mailbox_t</code> object
in	<i>msg</i>	the message to be posted on the mailbox
in	<i>timeout</i>	the number of ticks before the operation timeouts, the following special values are allowed: <ul style="list-style-type: none"> • <i>TIME_IMMEDIATE</i> immediate timeout. • <i>TIME_INFINITE</i> no timeout.

Returns

The operation status.

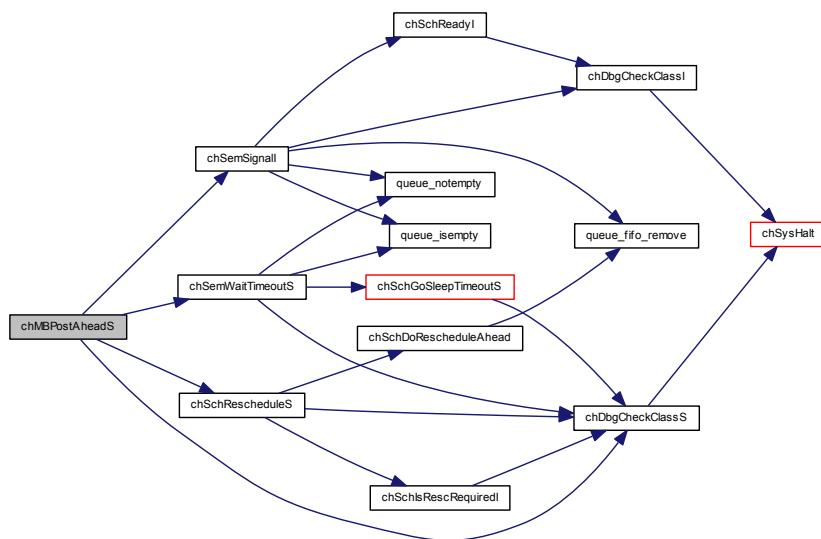
Return values

<i>MSG_OK</i>	if a message has been correctly posted.
<i>MSG_RESET</i>	if the mailbox has been reset while waiting.
<i>MSG_TIMEOUT</i>	if the operation has timed out.

Function Class:

This is an **S-Class** API, this function can be invoked from within a system lock zone by threads only.

Here is the call graph for this function:

**9.17.3.9 msg_t chMBPostAheadI (mailbox_t * mbp, msg_t msg)**

Posts an high priority message into a mailbox.

This variant is non-blocking, the function returns a timeout condition if the queue is full.

Parameters

in	<i>mbp</i>	the pointer to an initialized mailbox_t object
in	<i>msg</i>	the message to be posted on the mailbox

Returns

The operation status.

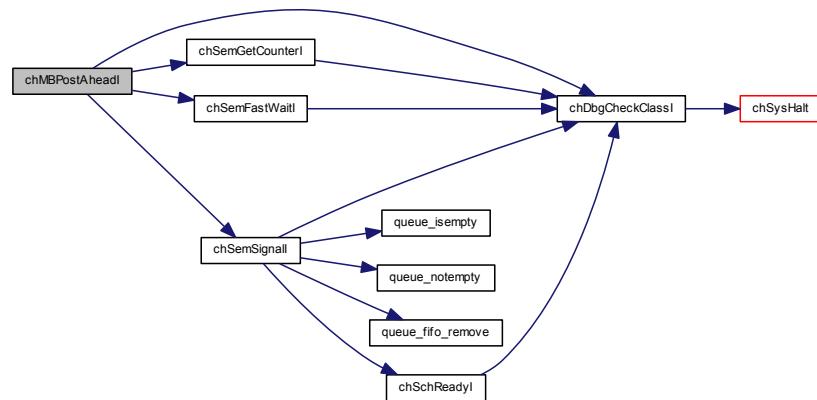
Return values

<i>MSG_OK</i>	if a message has been correctly posted.
<i>MSG_TIMEOUT</i>	if the mailbox is full and the message cannot be posted.

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:



9.17.3.10 `msg_t chMBFetch (mailbox_t *mbp, msg_t *msgp, systime_t timeout)`

Retrieves a message from a mailbox.

The invoking thread waits until a message is posted in the mailbox or the specified time runs out.

Parameters

in	<code>mbp</code>	the pointer to an initialized <code>mailbox_t</code> object
out	<code>msgp</code>	pointer to a message variable for the received message
in	<code>timeout</code>	the number of ticks before the operation timeouts, the following special values are allowed: <ul style="list-style-type: none"> • <code>TIME_IMMEDIATE</code> immediate timeout. • <code>TIME_INFINITE</code> no timeout.

Returns

The operation status.

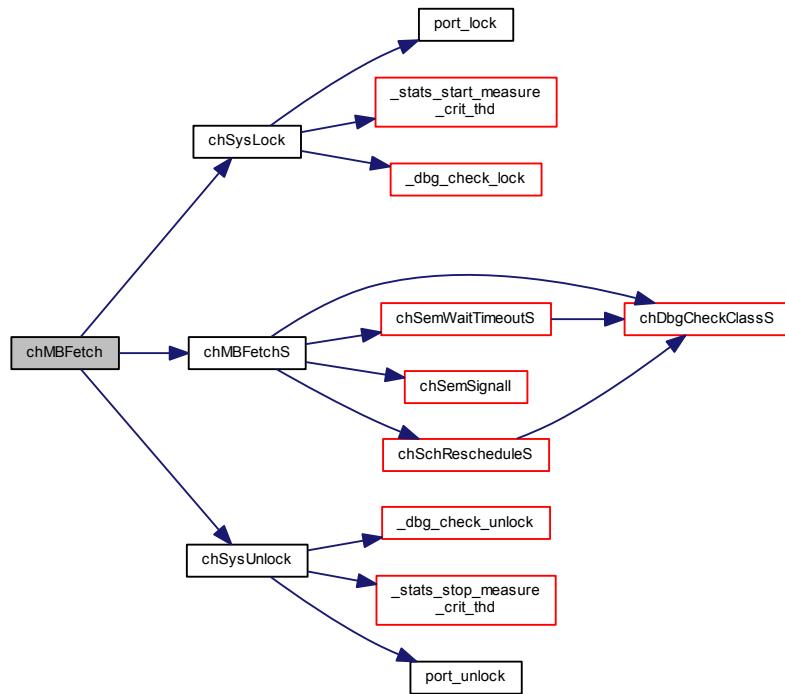
Return values

<code>MSG_OK</code>	if a message has been correctly fetched.
<code>MSG_RESET</code>	if the mailbox has been reset while waiting.
<code>MSG_TIMEOUT</code>	if the operation has timed out.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



9.17.3.11 `msg_t chMBFetchS (mailbox_t * mbp, msg_t * msgp, systime_t timeout)`

Retrieves a message from a mailbox.

The invoking thread waits until a message is posted in the mailbox or the specified time runs out.

Parameters

in	<code>mbp</code>	the pointer to an initialized <code>mailbox_t</code> object
out	<code>msgp</code>	pointer to a message variable for the received message
in	<code>timeout</code>	the number of ticks before the operation timeouts, the following special values are allowed: <ul style="list-style-type: none"> • <code>TIME_IMMEDIATE</code> immediate timeout. • <code>TIME_INFINITE</code> no timeout.

Returns

The operation status.

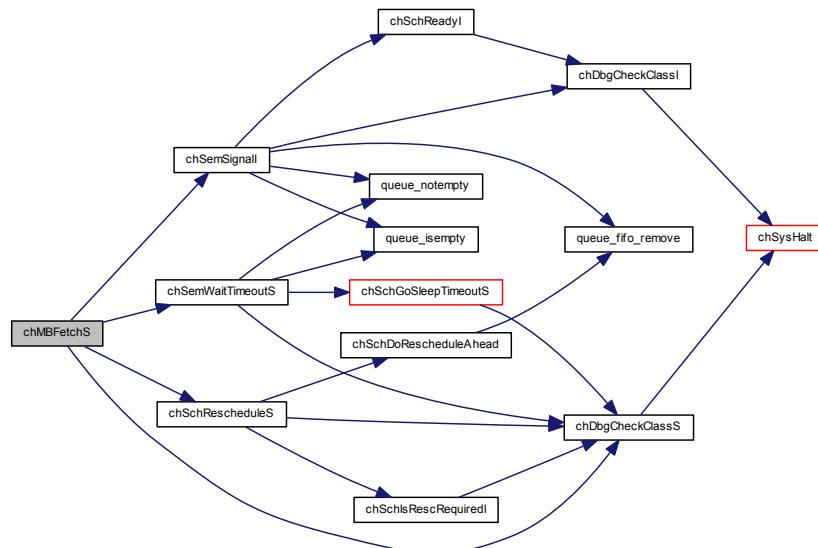
Return values

<code>MSG_OK</code>	if a message has been correctly fetched.
<code>MSG_RESET</code>	if the mailbox has been reset while waiting.
<code>MSG_TIMEOUT</code>	if the operation has timed out.

Function Class:

This is an **S-Class** API, this function can be invoked from within a system lock zone by threads only.

Here is the call graph for this function:



9.17.3.12 `msg_t chMBFetchI (mailbox_t *mbp, msg_t *msgp)`

Retrieves a message from a mailbox.

This variant is non-blocking, the function returns a timeout condition if the queue is empty.

Parameters

<code>in</code>	<code>mbp</code>	the pointer to an initialized <code>mailbox_t</code> object
<code>out</code>	<code>msgp</code>	pointer to a message variable for the received message

Returns

The operation status.

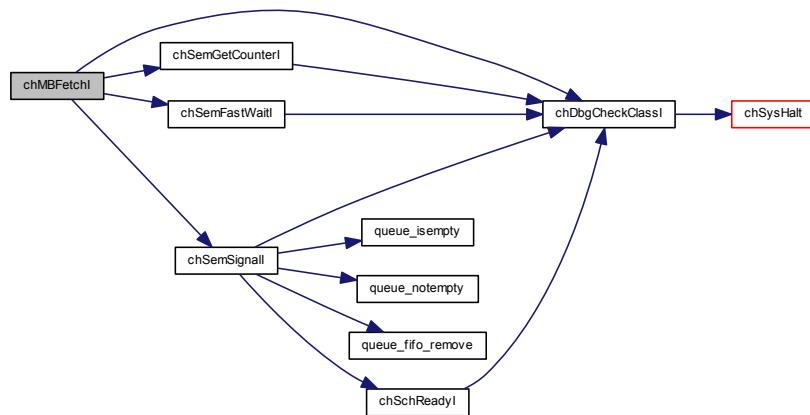
Return values

<code>MSG_OK</code>	if a message has been correctly fetched.
<code>MSG_TIMEOUT</code>	if the mailbox is empty and a message cannot be fetched.

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:



9.17.3.13 static size_t chMBGetSize(mailbox_t * mbp) [inline], [static]

Returns the mailbox buffer size.

Parameters

in	<code>mbp</code>	the pointer to an initialized <code>mailbox_t</code> object
----	------------------	---

Returns

The size of the mailbox.

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

9.17.3.14 static cnt_t chMBGetFreeCount(mailbox_t * mbp) [inline], [static]

Returns the number of free message slots into a mailbox.

Note

Can be invoked in any system state but if invoked out of a locked state then the returned value may change after reading.

The returned value can be less than zero when there are waiting threads on the internal semaphore.

Parameters

in	<code>mbp</code>	the pointer to an initialized <code>mailbox_t</code> object
----	------------------	---

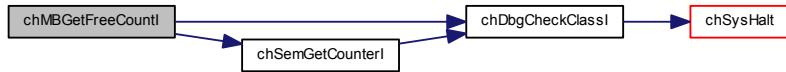
Returns

The number of empty message slots.

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:



9.17.3.15 static cnt_t chMBGetUsedCountl(mailbox_t * mbp) [inline], [static]

Returns the number of used message slots into a mailbox.

Note

Can be invoked in any system state but if invoked out of a locked state then the returned value may change after reading.

The returned value can be less than zero when there are waiting threads on the internal semaphore.

Parameters

in	<i>mbp</i>	the pointer to an initialized mailbox_t object
----	------------	--

Returns

The number of queued messages.

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:



9.17.3.16 static msg_t chMBPeekl(mailbox_t * mbp) [inline], [static]

Returns the next message in the queue without removing it.

Precondition

A message must be waiting in the queue for this function to work or it would return garbage. The correct way to use this macro is to use `chMBGetFullCountI()` and then use this macro, all within a lock state.

Parameters

in	<i>mbp</i>	the pointer to an initialized <code>mailbox_t</code> object
----	------------	---

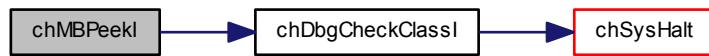
Returns

The next message in queue.

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:



9.18 I/O Queues

9.18.1 Detailed Description

ChibiOS/RT queues are mostly used in serial-like device drivers. The device drivers are usually designed to have a lower side (lower driver, it is usually an interrupt service routine) and an upper side (upper driver, accessed by the application threads).

There are several kind of queues:

- **Input queue**, unidirectional queue where the writer is the lower side and the reader is the upper side.
- **Output queue**, unidirectional queue where the writer is the upper side and the reader is the lower side.
- **Full duplex queue**, bidirectional queue. Full duplex queues are implemented by pairing an input queue and an output queue together.

Precondition

In order to use the I/O queues the CH_CFG_USE_QUEUES option must be enabled in `chconf.h`.

Macros

- `#define _INPUTQUEUE_DATA(name, buffer, size, inotify, link)`
Data part of a static input queue initializer.
- `#define INPUTQUEUE_DECL(name, buffer, size, inotify, link) input_queue_t name = _INPUTQUEUE_DATA(name, buffer, size, inotify, link)`
Static input queue initializer.
- `#define _OUTPUTQUEUE_DATA(name, buffer, size, onotify, link)`
Data part of a static output queue initializer.
- `#define OUTPUTQUEUE_DECL(name, buffer, size, onotify, link) output_queue_t name = _OUTPUTQUEUE_DATA(name, buffer, size, onotify, link)`
Static output queue initializer.

Queue functions returned status value

- `#define Q_OK MSG_OK`
Operation successful.
- `#define Q_TIMEOUT MSG_TIMEOUT`
Timeout condition.
- `#define Q_RESET MSG_RESET`
Queue has been reset.
- `#define Q_EMPTY (msg_t)-3`
Queue empty.
- `#define Q_FULL (msg_t)-4`
Queue full.,

Macro Functions

- `#define chQSizeX(qp)`
Returns the queue's buffer size.
- `#define chQSpaceI(qp) ((qp)->q_counter)`
Queue space.
- `#define chQGetLinkX(qp) ((qp)->q_link)`
Returns the queue application-defined link.

Typedefs

- **typedef struct io_queue io_queue_t**
Type of a generic I/O queue structure.
- **typedef void(* qnotify_t) (io_queue_t *qp)**
Queue notification callback type.
- **typedef io_queue_t input_queue_t**
Type of an input queue structure.
- **typedef io_queue_t output_queue_t**
Type of an output queue structure.

Data Structures

- **struct io_queue**
Generic I/O queue structure.

Functions

- **void chIQObjectInit (input_queue_t *iqp, uint8_t *bp, size_t size, qnotify_t infy, void *link)**
Initializes an input queue.
- **void chIQResetl (input_queue_t *iqp)**
Resets an input queue.
- **msg_t chIQPutl (input_queue_t *iqp, uint8_t b)**
Input queue write.
- **msg_t chIQGetTimeout (input_queue_t *iqp, systime_t timeout)**
Input queue read with timeout.
- **size_t chIQReadTimeout (input_queue_t *iqp, uint8_t *bp, size_t n, systime_t timeout)**
Input queue read with timeout.
- **void chOQObjectInit (output_queue_t *oqp, uint8_t *bp, size_t size, qnotify_t onfy, void *link)**
Initializes an output queue.
- **void chOQResetl (output_queue_t *oqp)**
Resets an output queue.
- **msg_t chOQPutTimeout (output_queue_t *oqp, uint8_t b, systime_t timeout)**
Output queue write with timeout.
- **msg_t chOQGetl (output_queue_t *oqp)**
Output queue read.
- **size_t chOQWriteTimeout (output_queue_t *oqp, const uint8_t *bp, size_t n, systime_t timeout)**
Output queue write with timeout.
- **static size_t chIQGetFulll (input_queue_t *iqp)**
Returns the filled space into an input queue.
- **static size_t chIQGetEmptyl (input_queue_t *iqp)**
Returns the empty space into an input queue.
- **static bool chIQIsEmptyl (input_queue_t *iqp)**
Evaluates to true if the specified input queue is empty.
- **static bool chIQIsFulll (input_queue_t *iqp)**
Evaluates to true if the specified input queue is full.
- **static msg_t chIQGet (input_queue_t *iqp)**
Input queue read.
- **static size_t chOQGetFulll (output_queue_t *oqp)**
Returns the filled space into an output queue.

- static size_t `chOQGetEmpty (output_queue_t *oqp)`
Returns the empty space into an output queue.
- static bool `chOQIsEmpty (output_queue_t *oqp)`
Evaluates to true if the specified output queue is empty.
- static bool `chOQIsFull (output_queue_t *oqp)`
Evaluates to true if the specified output queue is full.
- static msg_t `chOQPut (output_queue_t *oqp, uint8_t b)`
Output queue write.

9.18.2 Macro Definition Documentation

9.18.2.1 #define Q_OK MSG_OK

Operation successful.

9.18.2.2 #define Q_TIMEOUT MSG_TIMEOUT

Timeout condition.

9.18.2.3 #define Q_RESET MSG_RESET

Queue has been reset.

9.18.2.4 #define Q_EMPTY (msg_t)-3

Queue empty.

9.18.2.5 #define Q_FULL (msg_t)-4

Queue full.,

9.18.2.6 #define _INPUTQUEUE_DATA(name, buffer, size, inotify, link)

Value:

```
{
    \
    _THREADS_QUEUE_DATA (name),
    0U,
    (uint8_t *) (buffer),
    (uint8_t *) (buffer) + (size),
    (uint8_t *) (buffer),
    (uint8_t *) (buffer),
    (inotify),
    (link)
}
```



Data part of a static input queue initializer.

This macro should be used when statically initializing an input queue that is part of a bigger structure.

Parameters

in	<i>name</i>	the name of the input queue variable
in	<i>buffer</i>	pointer to the queue buffer area
in	<i>size</i>	size of the queue buffer area
in	<i>inotify</i>	input notification callback pointer
in	<i>link</i>	application defined pointer

```
9.18.2.7 #define INPUTQUEUE_DECL( name, buffer, size, inotify, link ) input_queue_t name =
    _INPUTQUEUE_DATA(name, buffer, size, inotify, link)
```

Static input queue initializer.

Statically initialized input queues require no explicit initialization using `chIQInit()`.

Parameters

in	<i>name</i>	the name of the input queue variable
in	<i>buffer</i>	pointer to the queue buffer area
in	<i>size</i>	size of the queue buffer area
in	<i>inotify</i>	input notification callback pointer
in	<i>link</i>	application defined pointer

```
9.18.2.8 #define _OUTPUTQUEUE_DATA( name, buffer, size, onotify, link )
```

Value:

```
{
    \
    _THREADS_QUEUE_DATA(name),
    (size),
    (uint8_t *)(buffer),
    (uint8_t *)(buffer) + (size),
    (uint8_t *)(buffer),
    (uint8_t *)(buffer),
    (onotify),
    (link)
}
```

Data part of a static output queue initializer.

This macro should be used when statically initializing an output queue that is part of a bigger structure.

Parameters

in	<i>name</i>	the name of the output queue variable
in	<i>buffer</i>	pointer to the queue buffer area
in	<i>size</i>	size of the queue buffer area
in	<i>onotify</i>	output notification callback pointer
in	<i>link</i>	application defined pointer

```
9.18.2.9 #define OUTPUTQUEUE_DECL( name, buffer, size, onotify, link ) output_queue_t name =
    _OUTPUTQUEUE_DATA(name, buffer, size, onotify, link)
```

Static output queue initializer.

Statically initialized output queues require no explicit initialization using `chOQInit()`.

Parameters

in	<i>name</i>	the name of the output queue variable
in	<i>buffer</i>	pointer to the queue buffer area
in	<i>size</i>	size of the queue buffer area
in	<i>onotify</i>	output notification callback pointer
in	<i>link</i>	application defined pointer

9.18.2.10 #define chQSizeX(*qp*)

Value:

```
/*lint -save -e9033 [10.8] The cast is safe.*/
((size_t)((qp)->q_top - (qp)->q_buffer)) \
/*lint -restore*/\ \
```

Returns the queue's buffer size.

Parameters

in	<i>qp</i>	pointer to a <code>io_queue_t</code> structure
----	-----------	--

Returns

The buffer size.

Function Class:

This is an **X-Class** API, this function can be invoked from any context.

9.18.2.11 #define chQSpaceI(*qp*) ((qp)->q_counter)

Queue space.

Returns the used space if used on an input queue or the empty space if used on an output queue.

Parameters

in	<i>qp</i>	pointer to a <code>io_queue_t</code> structure
----	-----------	--

Returns

The buffer space.

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

9.18.2.12 #define chQGetLinkX(*qp*) ((qp)->q_link)

Returns the queue application-defined link.

Parameters

in	<i>qp</i>	pointer to a <code>io_queue_t</code> structure
----	-----------	--

Returns

The application-defined link.

Function Class:

This is an **X-Class** API, this function can be invoked from any context.

9.18.3 Typedef Documentation

9.18.3.1 `typedef struct io_queue io_queue_t`

Type of a generic I/O queue structure.

9.18.3.2 `typedef void(* qnotify_t)(io_queue_t *qp)`

Queue notification callback type.

9.18.3.3 `typedef io_queue_t input_queue_t`

Type of an input queue structure.

This structure represents a generic asymmetrical input queue. Writing to the queue is non-blocking and can be performed from interrupt handlers or from within a kernel lock zone (see **I-Locked** and **S-Locked** states in [System States](#)). Reading the queue can be a blocking operation and is supposed to be performed by a system thread.

9.18.3.4 `typedef io_queue_t output_queue_t`

Type of an output queue structure.

This structure represents a generic asymmetrical output queue. Reading from the queue is non-blocking and can be performed from interrupt handlers or from within a kernel lock zone (see **I-Locked** and **S-Locked** states in [System States](#)). Writing the queue can be a blocking operation and is supposed to be performed by a system thread.

9.18.4 Function Documentation

9.18.4.1 `void chIQObjectInit(input_queue_t * iqp, uint8_t * bp, size_t size, qnotify_t infy, void * link)`

Initializes an input queue.

A Semaphore is internally initialized and works as a counter of the bytes contained in the queue.

Note

The callback is invoked from within the S-Locked system state, see [System States](#).

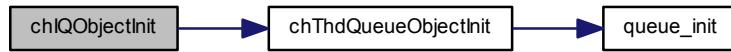
Parameters

<code>out</code>	<code>iqp</code>	pointer to an <code>input_queue_t</code> structure
<code>in</code>	<code>bp</code>	pointer to a memory area allocated as queue buffer
<code>in</code>	<code>size</code>	size of the queue buffer
<code>in</code>	<code>infy</code>	pointer to a callback function that is invoked when data is read from the queue. The value can be <code>NULL</code> .
<code>in</code>	<code>link</code>	application defined pointer

Function Class:

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

Here is the call graph for this function:



9.18.4.2 void chIQResetl (input_queue_t * iqp)

Resets an input queue.

All the data in the input queue is erased and lost, any waiting thread is resumed with status Q_RESET.

Note

A reset operation can be used by a low level driver in order to obtain immediate attention from the high level layers.

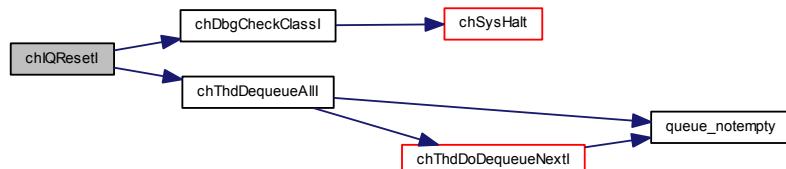
Parameters

in	<i>iqp</i>	pointer to an <code>input_queue_t</code> structure
----	------------	--

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:



9.18.4.3 msg_t chQPutl (input_queue_t * iqp, uint8_t b)

Input queue write.

A byte value is written into the low end of an input queue.

Parameters

in	<i>iqp</i>	pointer to an <code>input_queue_t</code> structure
in	<i>b</i>	the byte value to be written in the queue

Returns

The operation status.

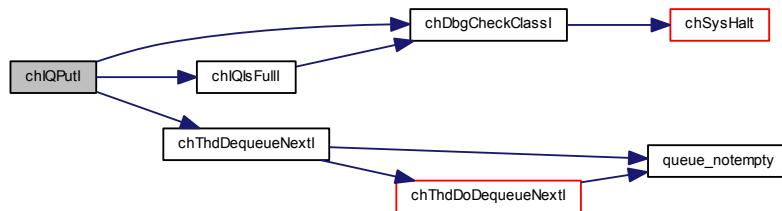
Return values

<i>Q_OK</i>	if the operation has been completed with success.
<i>Q_FULL</i>	if the queue is full and the operation cannot be completed.

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:

**9.18.4.4 msg_t chIQGetTimeout(input_queue_t * iqp, systime_t timeout)**

Input queue read with timeout.

This function reads a byte value from an input queue. If the queue is empty then the calling thread is suspended until a byte arrives in the queue or a timeout occurs.

Note

The callback is invoked before reading the character from the buffer or before entering the state `CH_STAT←E_WTQUEUE`.

Parameters

in	<i>iqp</i>	pointer to an <code>input_queue_t</code> structure
in	<i>timeout</i>	the number of ticks before the operation timeouts, the following special values are allowed: <ul style="list-style-type: none"> • <code>TIME_IMMEDIATE</code> immediate timeout. • <code>TIME_INFINITE</code> no timeout.

Returns

A byte value from the queue.

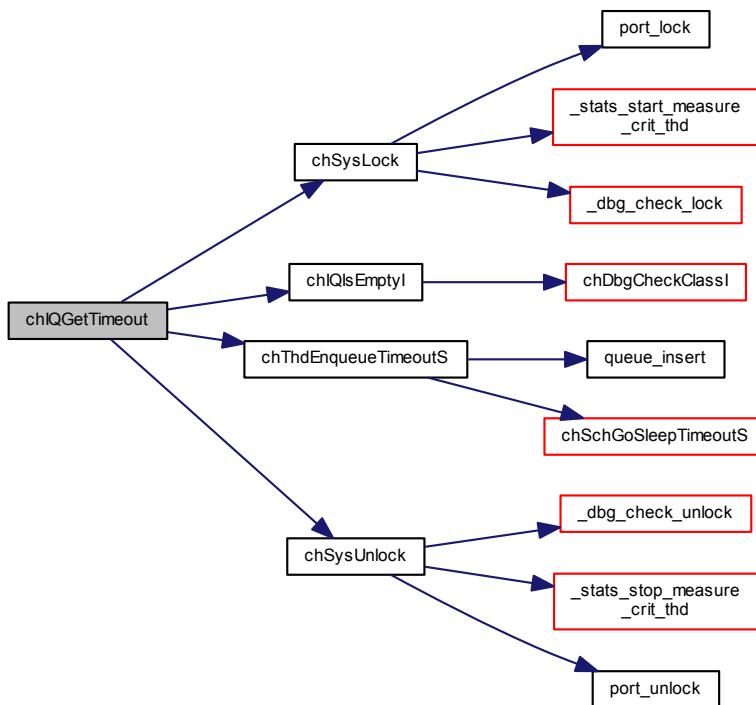
Return values

<i>Q_TIMEOUT</i>	if the specified time expired.
<i>Q_RESET</i>	if the queue has been reset.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



9.18.4.5 size_t chIQReadTimeout (input_queue_t * iqp, uint8_t * bp, size_t n, systime_t timeout)

Input queue read with timeout.

The function reads data from an input queue into a buffer. The operation completes when the specified amount of data has been transferred or after the specified timeout or if the queue has been reset.

Note

The function is not atomic, if you need atomicity it is suggested to use a semaphore or a mutex for mutual exclusion.

The callback is invoked before reading each character from the buffer or before entering the state CH_STA←TE_WTQUEUE.

Parameters

in	<i>iqp</i>	pointer to an <code>input_queue_t</code> structure
out	<i>bp</i>	pointer to the data buffer
in	<i>n</i>	the maximum amount of data to be transferred, the value 0 is reserved
in	<i>timeout</i>	the number of ticks before the operation timeouts, the following special values are allowed: <ul style="list-style-type: none"> • <code>TIME_IMMEDIATE</code> immediate timeout. • <code>TIME_INFINITE</code> no timeout.

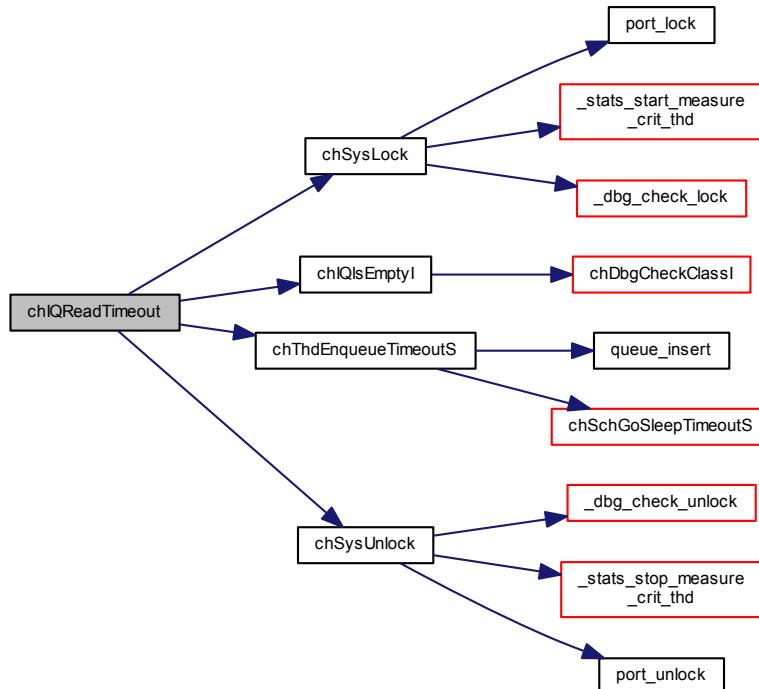
Returns

The number of bytes effectively transferred.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



9.18.4.6 void chOQObjectInit(`output_queue_t * oqp`, `uint8_t * bp`, `size_t size`, `qnotify_t onfy`, `void * link`)

Initializes an output queue.

A Semaphore is internally initialized and works as a counter of the free bytes in the queue.

Note

The callback is invoked from within the S-Locked system state, see [System States](#).

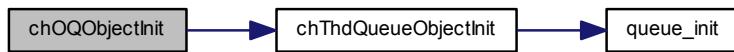
Parameters

<i>out</i>	<i>oqp</i>	pointer to an <code>output_queue_t</code> structure
<i>in</i>	<i>bp</i>	pointer to a memory area allocated as queue buffer
<i>in</i>	<i>size</i>	size of the queue buffer
<i>in</i>	<i>only</i>	pointer to a callback function that is invoked when data is written to the queue. The value can be <code>NULL</code> .
<i>in</i>	<i>link</i>	application defined pointer

Function Class:

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

Here is the call graph for this function:

**9.18.4.7 void chOQResetl (`output_queue_t` * *oqp*)**

Resets an output queue.

All the data in the output queue is erased and lost, any waiting thread is resumed with status `Q_RESET`.

Note

A reset operation can be used by a low level driver in order to obtain immediate attention from the high level layers.

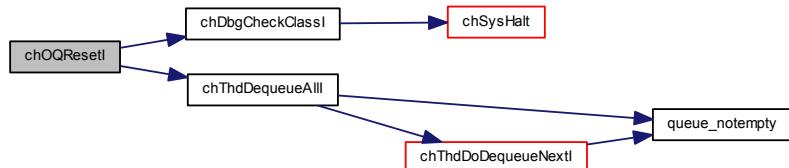
Parameters

<i>in</i>	<i>oqp</i>	pointer to an <code>output_queue_t</code> structure
-----------	------------	---

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:



9.18.4.8 `msg_t chOQPutTimeout(output_queue_t *oqp, uint8_t b, systime_t timeout)`

Output queue write with timeout.

This function writes a byte value to an output queue. If the queue is full then the calling thread is suspended until there is space in the queue or a timeout occurs.

Note

The callback is invoked after writing the character into the buffer.

Parameters

in	<i>oqp</i>	pointer to an <code>output_queue_t</code> structure
in	<i>b</i>	the byte value to be written in the queue
in	<i>timeout</i>	the number of ticks before the operation timeouts, the following special values are allowed: <ul style="list-style-type: none"> • <code>TIME_IMMEDIATE</code> immediate timeout. • <code>TIME_INFINITE</code> no timeout.

Returns

The operation status.

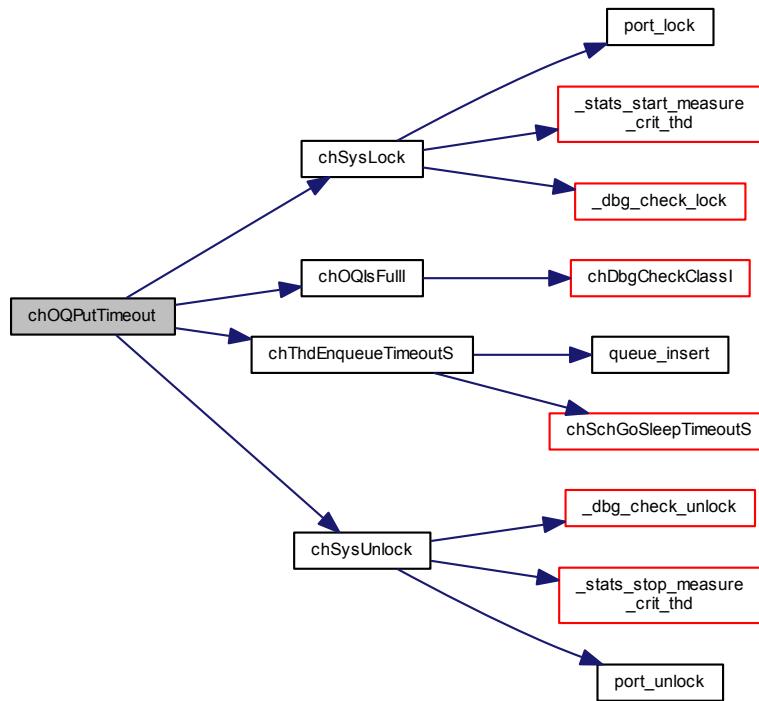
Return values

<code>Q_OK</code>	if the operation succeeded.
<code>Q_TIMEOUT</code>	if the specified time expired.
<code>Q_RESET</code>	if the queue has been reset.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



9.18.4.9 `msg_t chOQGet(output_queue_t * oqp)`

Output queue read.

A byte value is read from the low end of an output queue.

Parameters

in	<code>oqp</code>	pointer to an <code>output_queue_t</code> structure
----	------------------	---

Returns

The byte value from the queue.

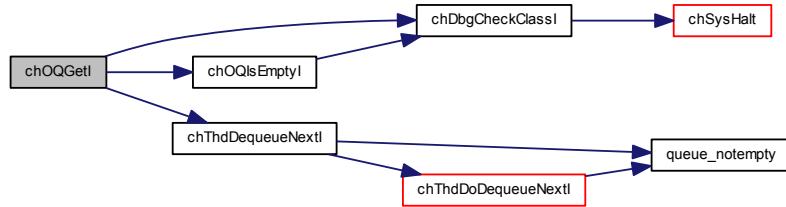
Return values

<code>Q_EMPTY</code>	if the queue is empty.
----------------------	------------------------

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:



9.18.4.10 size_t chOQWriteTimeout (output_queue_t * oqp, const uint8_t * bp, size_t n, systime_t timeout)

Output queue write with timeout.

The function writes data from a buffer to an output queue. The operation completes when the specified amount of data has been transferred or after the specified timeout or if the queue has been reset.

Note

The function is not atomic, if you need atomicity it is suggested to use a semaphore or a mutex for mutual exclusion.

The callback is invoked after writing each character into the buffer.

Parameters

in	<i>oqp</i>	pointer to an <code>output_queue_t</code> structure
in	<i>bp</i>	pointer to the data buffer
in	<i>n</i>	the maximum amount of data to be transferred, the value 0 is reserved
in	<i>timeout</i>	the number of ticks before the operation timeouts, the following special values are allowed: <ul style="list-style-type: none"> • <code>TIME_IMMEDIATE</code> immediate timeout. • <code>TIME_INFINITE</code> no timeout.

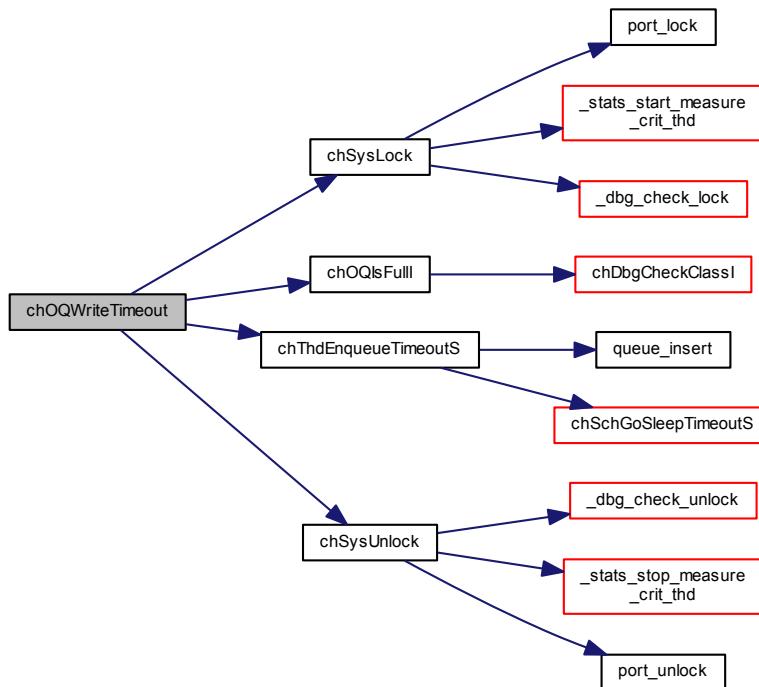
Returns

The number of bytes effectively transferred.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



9.18.4.11 static size_t chIQGetFull(input_queue_t * iqp) [inline], [static]

Returns the filled space into an input queue.

Parameters

in	<i>iqp</i>	pointer to an <code>input_queue_t</code> structure
----	------------	--

Returns

The number of full bytes in the queue.

Return values

<i>O</i>	if the queue is empty.
----------	------------------------

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:



9.18.4.12 static size_t chIQGetEmptyI (input_queue_t * iqp) [inline], [static]

Returns the empty space into an input queue.

Parameters

in	<i>iqp</i>	pointer to an <code>input_queue_t</code> structure
----	------------	--

Returns

The number of empty bytes in the queue.

Return values

0	if the queue is full.
---	-----------------------

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:



9.18.4.13 static bool chIQIsEmptyI (input_queue_t * iqp) [inline], [static]

Evaluates to `true` if the specified input queue is empty.

Parameters

in	<i>iqp</i>	pointer to an <code>input_queue_t</code> structure
----	------------	--

Returns

The queue status.

Return values

<i>false</i>	if the queue is not empty.
<i>true</i>	if the queue is empty.

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:



9.18.4.14 static bool chIqIsFullI (input_queue_t * iqp) [inline], [static]

Evaluates to `true` if the specified input queue is full.

Parameters

in	<i>iqp</i>	pointer to an <code>input_queue_t</code> structure
----	------------	--

Returns

The queue status.

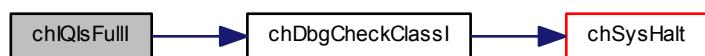
Return values

<i>false</i>	if the queue is not full.
<i>true</i>	if the queue is full.

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:



9.18.4.15 static msg_t chIQGet(input_queue_t * iqp) [inline], [static]

Input queue read.

This function reads a byte value from an input queue. If the queue is empty then the calling thread is suspended until a byte arrives in the queue.

Parameters

in	<i>iqp</i>	pointer to an <code>input_queue_t</code> structure
----	------------	--

Returns

A byte value from the queue.

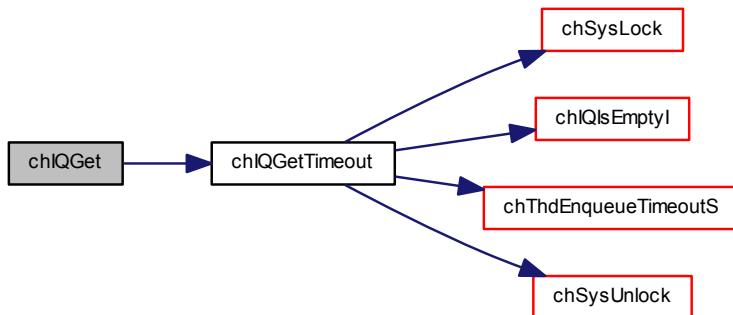
Return values

<i>Q_RESET</i>	if the queue has been reset.
----------------	------------------------------

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



9.18.4.16 static size_t chOQGetFull(output_queue_t * oqp) [inline], [static]

Returns the filled space into an output queue.

Parameters

in	<i>oqp</i>	pointer to an <code>output_queue_t</code> structure
----	------------	---

Returns

The number of full bytes in the queue.

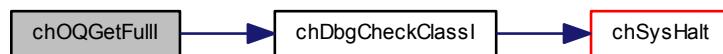
Return values

<i>O</i>	if the queue is empty.
----------	------------------------

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:



9.18.4.17 static size_t chOQGetEmptyl(output_queue_t * oqp) [inline], [static]

Returns the empty space into an output queue.

Parameters

in	<i>oqp</i>	pointer to an <code>output_queue_t</code> structure
----	------------	---

Returns

The number of empty bytes in the queue.

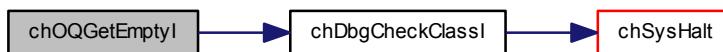
Return values

<i>O</i>	if the queue is full.
----------	-----------------------

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:



9.18.4.18 static bool chOQIsEmptyl(output_queue_t * oqp) [inline], [static]

Evaluates to `true` if the specified output queue is empty.

Parameters

in	<i>oqp</i>	pointer to an <code>output_queue_t</code> structure
----	------------	---

Returns

The queue status.

Return values

<i>false</i>	if the queue is not empty.
<i>true</i>	if the queue is empty.

Function Class:

This is an **I-Class API**, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:

**9.18.4.19 static bool chOQIsFull(`output_queue_t` * *oqp*) [inline], [static]**

Evaluates to `true` if the specified output queue is full.

Parameters

in	<i>oqp</i>	pointer to an <code>output_queue_t</code> structure
----	------------	---

Returns

The queue status.

Return values

<i>false</i>	if the queue is not full.
<i>true</i>	if the queue is full.

Function Class:

This is an **I-Class API**, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:



9.18.4.20 static msg_t chOQPut(output_queue_t * oqp, uint8_t b) [inline], [static]

Output queue write.

This function writes a byte value to an output queue. If the queue is full then the calling thread is suspended until there is space in the queue.

Parameters

in	<i>oqp</i>	pointer to an <code>output_queue_t</code> structure
in	<i>b</i>	the byte value to be written in the queue

Returns

The operation status.

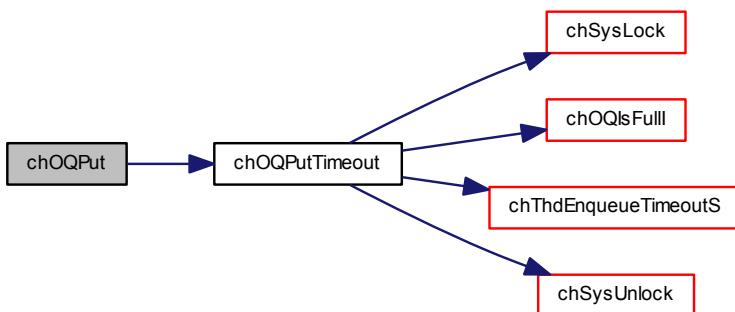
Return values

<code>Q_OK</code>	if the operation succeeded.
<code>Q_RESET</code>	if the queue has been reset.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



9.19 Memory Management

9.19.1 Detailed Description

Memory Management services.

Modules

- [Core Memory Manager](#)
- [Heaps](#)
- [Memory Pools](#)
- [Dynamic Threads](#)

9.20 Core Memory Manager

9.20.1 Detailed Description

Core Memory Manager related APIs and services.

Operation mode

The core memory manager is a simplified allocator that only allows to allocate memory blocks without the possibility to free them.

This allocator is meant as a memory blocks provider for the other allocators such as:

- C-Runtime allocator (through a compiler specific adapter module).
- Heap allocator (see [Heaps](#)).
- Memory pools allocator (see [Memory Pools](#)).

By having a centralized memory provider the various allocators can coexist and share the main memory.
This allocator, alone, is also useful for very simple applications that just require a simple way to get memory blocks.

Precondition

In order to use the core memory manager APIs the `CH_CFG_USE_MEMCORE` option must be enabled in `chconf.h`.

Alignment support macros

- `#define MEM_ALIGN_SIZE sizeof(stkalign_t)`
Alignment size constant.
- `#define MEM_ALIGN_MASK (MEM_ALIGN_SIZE - 1U)`
Alignment mask constant.
- `#define MEM_ALIGN_PREV(p) ((size_t)(p) & ~MEM_ALIGN_MASK)`
Alignment helper macro.
- `#define MEM_ALIGN_NEXT(p) MEM_ALIGN_PREV((size_t)(p) + MEM_ALIGN_MASK)`
Alignment helper macro.
- `#define MEM_IS_ALIGNED(p) (((size_t)(p) & MEM_ALIGN_MASK) == 0U)`
Returns whatever a pointer or memory size is aligned to the type `stkalign_t`.

Typedefs

- `typedef void *(*memgetfunc_t) (size_t size)`
Memory get function.

Functions

- `void _core_init (void)`
Low level memory manager initialization.
- `void * chCoreAlloc (size_t size)`
Allocates a memory block.
- `void * chCoreAlloc1 (size_t size)`
Allocates a memory block.
- `size_t chCoreGetStatusX (void)`
Core memory status.

9.20.2 Macro Definition Documentation

9.20.2.1 `#define MEM_ALIGN_SIZE sizeof(stkalign_t)`

Alignment size constant.

9.20.2.2 `#define MEM_ALIGN_MASK (MEM_ALIGN_SIZE - 1U)`

Alignment mask constant.

9.20.2.3 `#define MEM_ALIGN_PREV(p) ((size_t)(p) & ~MEM_ALIGN_MASK)`

Alignment helper macro.

9.20.2.4 `#define MEM_ALIGN_NEXT(p) MEM_ALIGN_PREV((size_t)(p) + MEM_ALIGN_MASK)`

Alignment helper macro.

9.20.2.5 `#define MEM_IS_ALIGNED(p) (((size_t)(p) & MEM_ALIGN_MASK) == 0U)`

Returns whatever a pointer or memory size is aligned to the type `stkalign_t`.

9.20.3 Typedef Documentation

9.20.3.1 `typedef void*(* memgetfunc_t)(size_t size)`

Memory get function.

9.20.4 Function Documentation

9.20.4.1 `void _core_init(void)`

Low level memory manager initialization.

Function Class:

Not an API, this function is for internal use only.

9.20.4.2 `void * chCoreAlloc(size_t size)`

Allocates a memory block.

The size of the returned block is aligned to the alignment type so it is not possible to allocate less than `MEM_ALIGN_SIZE`.

Parameters

in	size	the size of the block to be allocated
----	------	---------------------------------------

Returns

A pointer to the allocated memory block.

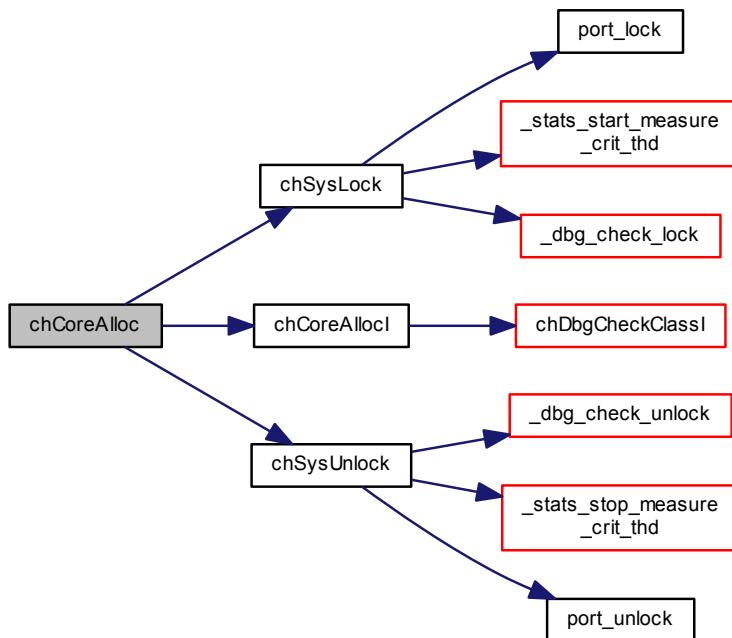
Return values

<code>NULL</code>	allocation failed, core memory exhausted.
-------------------	---

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



9.20.4.3 `void * chCoreAllocI (size_t size)`

Allocates a memory block.

The size of the returned block is aligned to the alignment type so it is not possible to allocate less than `MEM_ALIGN_SIZE`.

Parameters

<code>in</code>	<code>size</code>	the size of the block to be allocated.
-----------------	-------------------	--

Returns

A pointer to the allocated memory block.

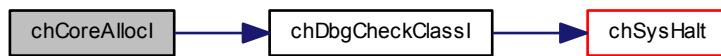
Return values

NULL	allocation failed, core memory exhausted.
------	---

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:

**9.20.4.4 size_t chCoreGetStatusX (void)**

Core memory status.

Returns

The size, in bytes, of the free core memory.

Function Class:

This is an **X-Class** API, this function can be invoked from any context.

9.21 Heaps

9.21.1 Detailed Description

Heap Allocator related APIs.

Operation mode

The heap allocator implements a first-fit strategy and its APIs are functionally equivalent to the usual `malloc()` and `free()` library functions. The main difference is that the OS heap APIs are guaranteed to be thread safe.

Precondition

In order to use the heap APIs the `CH_CFG_USE_HEAP` option must be enabled in `chconf.h`.

Typedefs

- `typedef struct memory_heap memory_heap_t`

Type of a memory heap.

Data Structures

- `union heap_header`
Memory heap block header.
- `struct memory_heap`

Structure describing a memory heap.

Functions

- `void _heap_init (void)`
Initializes the default heap.
- `void chHeapObjectInit (memory_heap_t *heapp, void *buf, size_t size)`
Initializes a memory heap from a static memory area.
- `void * chHeapAlloc (memory_heap_t *heapp, size_t size)`
Allocates a block of memory from the heap by using the first-fit algorithm.
- `void chHeapFree (void *p)`
Frees a previously allocated memory block.
- `size_t chHeapStatus (memory_heap_t *heapp, size_t *sizep)`
Reports the heap status.

Variables

- `static memory_heap_t default_heap`
Default heap descriptor.

9.21.2 Typedef Documentation

9.21.2.1 `typedef struct memory_heap memory_heap_t`

Type of a memory heap.

9.21.3 Function Documentation

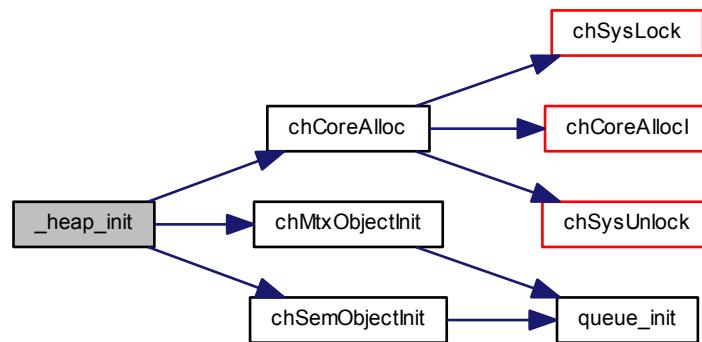
9.21.3.1 void _heap_init(void)

Initializes the default heap.

Function Class:

Not an API, this function is for internal use only.

Here is the call graph for this function:



9.21.3.2 void chHeapObjectInit(memory_heap_t * heapp, void * buf, size_t size)

Initializes a memory heap from a static memory area.

Precondition

Both the heap buffer base and the heap size must be aligned to the `stkalign_t` type size.

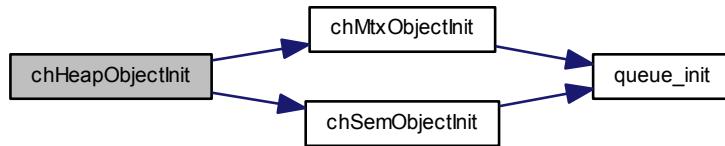
Parameters

<code>out</code>	<code>heapp</code>	pointer to the memory heap descriptor to be initialized
<code>in</code>	<code>buf</code>	heap buffer base
<code>in</code>	<code>size</code>	heap size

Function Class:

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

Here is the call graph for this function:

**9.21.3.3 void * chHeapAlloc (`memory_heap_t` * `heapp`, `size_t` `size`)**

Allocates a block of memory from the heap by using the first-fit algorithm.

The allocated block is guaranteed to be properly aligned for a pointer data type (`stkalign_t`).

Parameters

<code>in</code>	<code>heapp</code>	pointer to a heap descriptor or <code>NULL</code> in order to access the default heap.
<code>in</code>	<code>size</code>	the size of the block to be allocated. Note that the allocated block may be a bit bigger than the requested size for alignment and fragmentation reasons.

Returns

A pointer to the allocated block.

Return values

<code>NULL</code>	if the block cannot be allocated.
-------------------	-----------------------------------

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

9.21.3.4 void chHeapFree (`void` * `p`)

Frees a previously allocated memory block.

Parameters

in	<i>p</i>	pointer to the memory block to be freed
----	----------	---

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

9.21.3.5 size_t chHeapStatus (memory_heap_t * *heapp*, size_t * *sizep*)

Reports the heap status.

Note

This function is meant to be used in the test suite, it should not be really useful for the application code.

Parameters

in	<i>heapp</i>	pointer to a heap descriptor or <code>NULL</code> in order to access the default heap.
in	<i>sizep</i>	pointer to a variable that will receive the total fragmented free space

Returns

The number of fragments in the heap.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

9.21.4 Variable Documentation**9.21.4.1 memory_heap_t default_heap [static]**

Default heap descriptor.

9.22 Memory Pools

9.22.1 Detailed Description

Memory Pools related APIs and services.

Operation mode

The Memory Pools APIs allow to allocate/free fixed size objects in **constant time** and reliably without memory fragmentation problems.

Memory Pools do not enforce any alignment constraint on the contained object however the objects must be properly aligned to contain a pointer to void.

Precondition

In order to use the memory pools APIs the `CH_CFG_USE_MEMPOOLS` option must be enabled in `chconf.h`.

Macros

- `#define _MEMORYPOOL_DATA(name, size, provider) {NULL, size, provider}`
Data part of a static memory pool initializer.
- `#define MEMORYPOOL_DECL(name, size, provider) memory_pool_t name = _MEMORYPOOL_DATA(name, size, provider)`
Static memory pool initializer in hungry mode.

Data Structures

- struct `pool_header`
Memory pool free object header.
- struct `memory_pool_t`
Memory pool descriptor.

Functions

- `void chPoolObjectInit (memory_pool_t *mp, size_t size, memgetfunc_t provider)`
Initializes an empty memory pool.
- `void chPoolLoadArray (memory_pool_t *mp, void *p, size_t n)`
Loads a memory pool with an array of static objects.
- `void * chPoolAlloc (memory_pool_t *mp)`
Allocates an object from a memory pool.
- `void * chPoolAlloc (memory_pool_t *mp)`
Allocates an object from a memory pool.
- `void chPoolFree (memory_pool_t *mp, void *objp)`
Releases an object into a memory pool.
- `void chPoolFree (memory_pool_t *mp, void *objp)`
Releases an object into a memory pool.
- `static void chPoolAdd (memory_pool_t *mp, void *objp)`
Adds an object to a memory pool.
- `static void chPoolAddl (memory_pool_t *mp, void *objp)`
Adds an object to a memory pool.

9.22.2 Macro Definition Documentation

9.22.2.1 `#define _MEMORYPOOL_DATA(name, size, provider) {NULL, size, provider}`

Data part of a static memory pool initializer.

This macro should be used when statically initializing a memory pool that is part of a bigger structure.

Parameters

in	<i>name</i>	the name of the memory pool variable
in	<i>size</i>	size of the memory pool contained objects
in	<i>provider</i>	memory provider function for the memory pool

9.22.2.2 `#define MEMORYPOOL_DECL(name, size, provider) memory_pool_t name = _MEMORYPOOL_DATA(name, size, provider)`

Static memory pool initializer in hungry mode.

Statically initialized memory pools require no explicit initialization using `chPoolInit()`.

Parameters

in	<i>name</i>	the name of the memory pool variable
in	<i>size</i>	size of the memory pool contained objects
in	<i>provider</i>	memory provider function for the memory pool or <code>NULL</code> if the pool is not allowed to grow automatically

9.22.3 Function Documentation

9.22.3.1 `void chPoolObjectInit(memory_pool_t * mp, size_t size, memgetfunc_t provider)`

Initializes an empty memory pool.

Parameters

out	<i>mp</i>	pointer to a <code>memory_pool_t</code> structure
in	<i>size</i>	the size of the objects contained in this memory pool, the minimum accepted size is the size of a pointer to void.
in	<i>provider</i>	memory provider function for the memory pool or <code>NULL</code> if the pool is not allowed to grow automatically

Function Class:

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

9.22.3.2 `void chPoolLoadArray(memory_pool_t * mp, void * p, size_t n)`

Loads a memory pool with an array of static objects.

Precondition

The memory pool must be already been initialized.

The array elements must be of the right size for the specified memory pool.

Postcondition

The memory pool contains the elements of the input array.

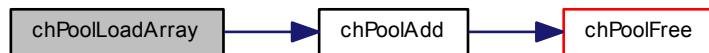
Parameters

in	<i>mp</i>	pointer to a memory_pool_t structure
in	<i>p</i>	pointer to the array first element
in	<i>n</i>	number of elements in the array

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

**9.22.3.3 void * chPoolAlloc (memory_pool_t * mp)**

Allocates an object from a memory pool.

Precondition

The memory pool must be already been initialized.

Parameters

in	<i>mp</i>	pointer to a memory_pool_t structure
----	-----------	--

Returns

The pointer to the allocated object.

Return values

<i>NULL</i>	if pool is empty.
-------------	-------------------

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:



9.22.3.4 void * chPoolAlloc (memory_pool_t * mp)

Allocates an object from a memory pool.

Precondition

The memory pool must be already been initialized.

Parameters

in	<i>mp</i>	pointer to a memory_pool_t structure
----	-----------	--

Returns

The pointer to the allocated object.

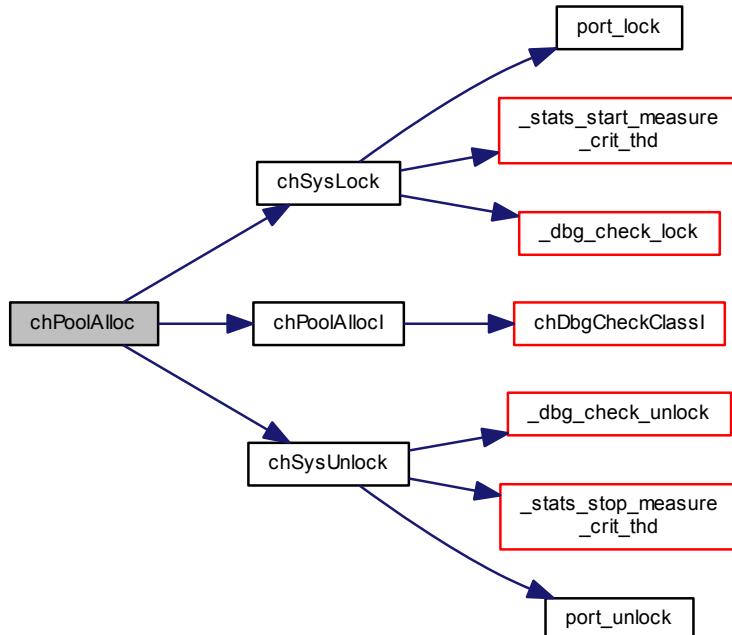
Return values

<i>NULL</i>	if pool is empty.
-------------	-------------------

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



9.22.3.5 void chPoolFree (memory_pool_t * mp, void * objp)

Releases an object into a memory pool.

Precondition

- The memory pool must be already been initialized.
- The freed object must be of the right size for the specified memory pool.
- The object must be properly aligned to contain a pointer to void.

Parameters

in	<i>mp</i>	pointer to a <code>memory_pool_t</code> structure
in	<i>objp</i>	the pointer to the object to be released

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:

**9.22.3.6 void chPoolFree (`memory_pool_t` * *mp*, void * *objp*)**

Releases an object into a memory pool.

Precondition

- The memory pool must be already been initialized.
- The freed object must be of the right size for the specified memory pool.
- The object must be properly aligned to contain a pointer to void.

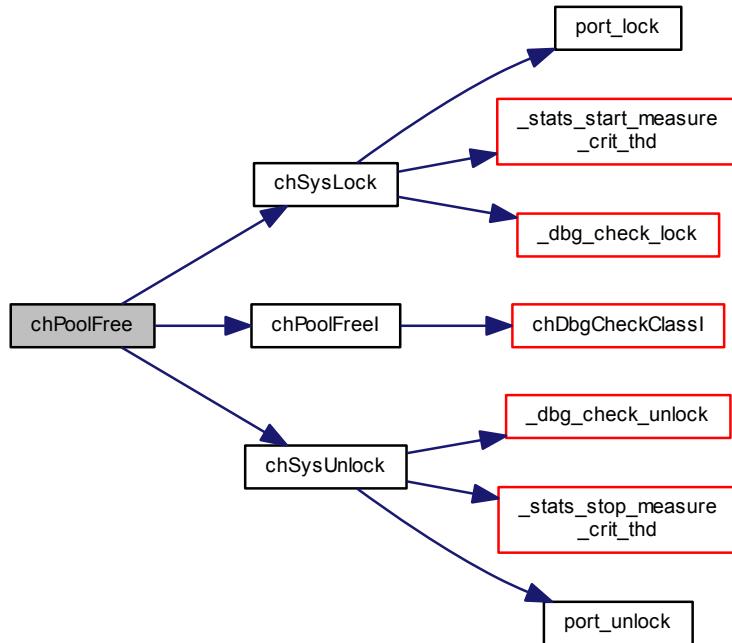
Parameters

in	<i>mp</i>	pointer to a <code>memory_pool_t</code> structure
in	<i>objp</i>	the pointer to the object to be released

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



9.22.3.7 static void chPoolAdd([memory_pool_t](#) * *mp*, void * *objp*) [inline], [static]

Adds an object to a memory pool.

Precondition

- The memory pool must be already been initialized.
- The added object must be of the right size for the specified memory pool.
- The added object must be memory aligned to the size of `stkalign_t` type.

Note

This function is just an alias for `chPoolFree()` and has been added for clarity.

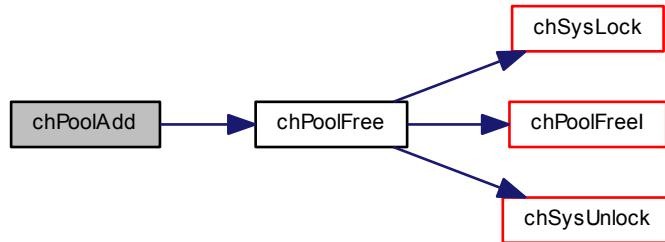
Parameters

in	<i>mp</i>	pointer to a memory_pool_t structure
in	<i>objp</i>	the pointer to the object to be added

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



9.22.3.8 static void chPoolAddl ([memory_pool_t](#) * *mp*, void * *objp*) [inline], [static]

Adds an object to a memory pool.

Precondition

- The memory pool must be already been initialized.
- The added object must be of the right size for the specified memory pool.
- The added object must be memory aligned to the size of `stkalign_t` type.

Note

This function is just an alias for [chPoolFree\(\)](#) and has been added for clarity.

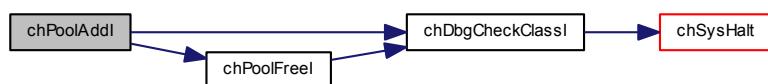
Parameters

in	<i>mp</i>	pointer to a memory_pool_t structure
in	<i>objp</i>	the pointer to the object to be added

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:



9.23 Dynamic Threads

9.23.1 Detailed Description

Dynamic threads related APIs and services.

Functions

- `thread_t * chThdAddRef (thread_t *tp)`

Adds a reference to a thread object.

- `void chThdRelease (thread_t *tp)`

Releases a reference to a thread object.

- `thread_t * chThdCreateFromHeap (memory_heap_t *heapp, size_t size, tprio_t prio, tfunc_t pf, void *arg)`

Creates a new thread allocating the memory from the heap.

- `thread_t * chThdCreateFromMemoryPool (memory_pool_t *mp, tprio_t prio, tfunc_t pf, void *arg)`

Creates a new thread allocating the memory from the specified memory pool.

9.23.2 Function Documentation

9.23.2.1 `thread_t * chThdAddRef (thread_t * tp)`

Adds a reference to a thread object.

Precondition

The configuration option `CH_CFG_USE_DYNAMIC` must be enabled in order to use this function.

Parameters

in	<code>tp</code>	pointer to the thread
----	-----------------	-----------------------

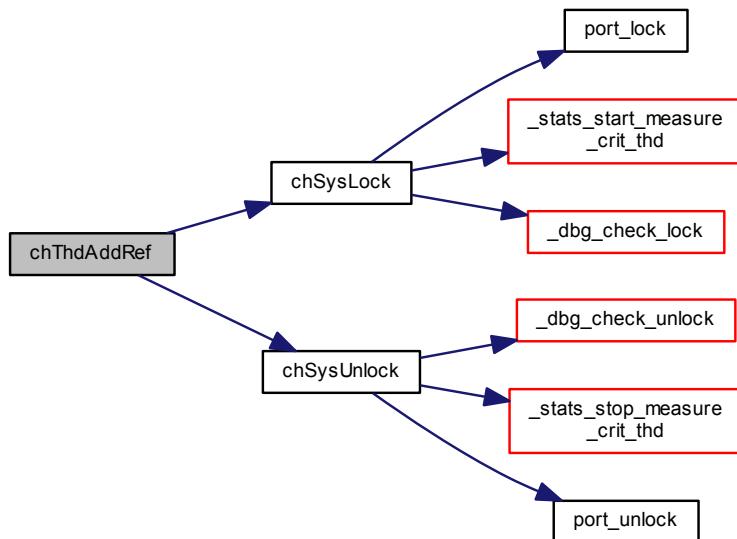
Returns

The same thread pointer passed as parameter representing the new reference.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

**9.23.2.2 void chThdRelease (`thread_t * tp`)**

Releases a reference to a thread object.

If the references counter reaches zero **and** the thread is in the `CH_STATE_FINAL` state then the thread's memory is returned to the proper allocator.

Precondition

The configuration option `CH_CFG_USE_DYNAMIC` must be enabled in order to use this function.

Note

Static threads are not affected.

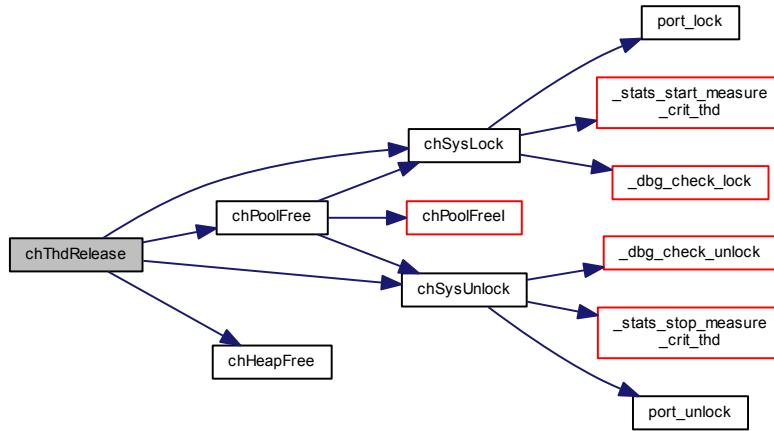
Parameters

<code>in</code>	<code>tp</code>	pointer to the thread
-----------------	-----------------	-----------------------

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



9.23.2.3 `thread_t * chThdCreateFromHeap (memory_heap_t * heapp, size_t size, tprio_t prio, tfunc_t pf, void * arg)`

Creates a new thread allocating the memory from the heap.

Precondition

The configuration options `CH_CFG_USE_DYNAMIC` and `CH_CFG_USE_HEAP` must be enabled in order to use this function.

Note

A thread can terminate by calling `chThdExit ()` or by simply returning from its main function. The memory allocated for the thread is not released when the thread terminates but when a `chThdWait ()` is performed.

Parameters

in	<code>heapp</code>	heap from which allocate the memory or <code>NULL</code> for the default heap
in	<code>size</code>	size of the working area to be allocated
in	<code>prio</code>	the priority level for the new thread
in	<code>pf</code>	the thread function
in	<code>arg</code>	an argument passed to the thread function. It can be <code>NULL</code> .

Returns

The pointer to the `thread_t` structure allocated for the thread into the working space area.

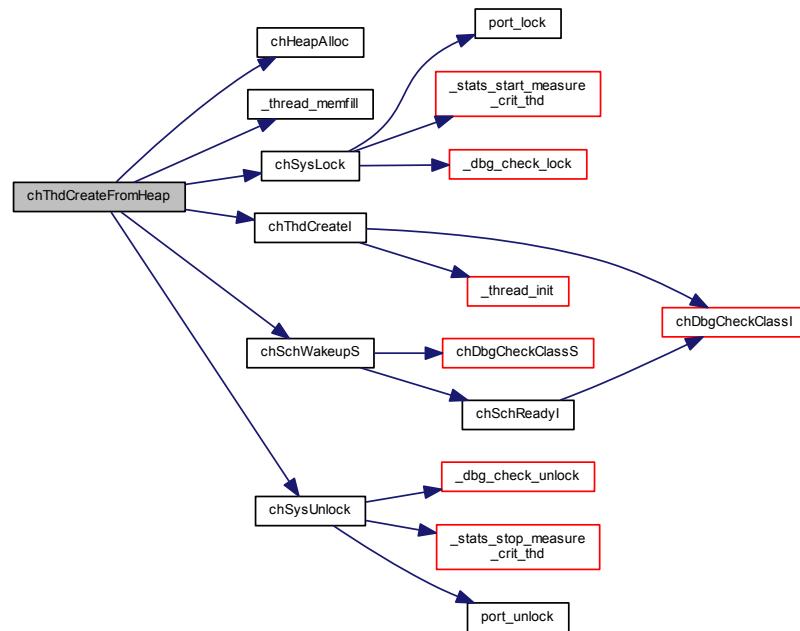
Return values

<code>NULL</code>	if the memory cannot be allocated.
-------------------	------------------------------------

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



9.23.2.4 `thread_t * chThdCreateFromMemoryPool (memory_pool_t * mp, tprio_t prio, tfunc_t pf, void * arg)`

Creates a new thread allocating the memory from the specified memory pool.

Precondition

The configuration options `CH_CFG_USE_DYNAMIC` and `CH_CFG_USE_MEMPOOLS` must be enabled in order to use this function.

Note

A thread can terminate by calling `chThdExit()` or by simply returning from its main function. The memory allocated for the thread is not released when the thread terminates but when a `chThdWait()` is performed.

Parameters

in	<code>mp</code>	pointer to the memory pool object
in	<code>prio</code>	the priority level for the new thread
in	<code>pf</code>	the thread function
in	<code>arg</code>	an argument passed to the thread function. It can be NULL.

Returns

The pointer to the `thread_t` structure allocated for the thread into the working space area.

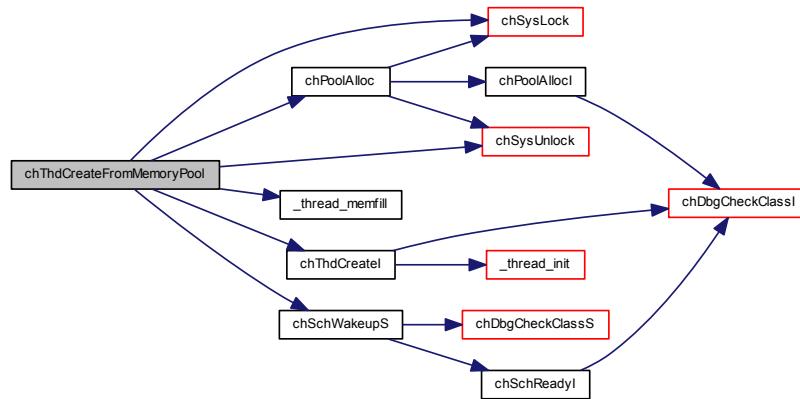
Return values

<code>NULL</code>	if the memory pool is empty.
-------------------	------------------------------

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



9.24 Streams and Files

9.24.1 Detailed Description

Stream and Files interfaces.

Modules

- [Abstract Sequential Streams](#)

9.25 Abstract Sequential Streams

9.25.1 Detailed Description

This module define an abstract interface for generic data streams. Note that no code is present, just abstract interfaces-like structures, you should look at the system as to a set of abstract C++ classes (even if written in C). This system has then advantage to make the access to data streams independent from the implementation logic. The stream interface can be used as base class for high level object types such as files, sockets, serial ports, pipes etc.

Macros

- `#define _base_sequential_stream_methods`
BaseSequentialStream specific methods.
- `#define _base_sequential_stream_data`
BaseSequentialStream specific data.

Macro Functions (BaseSequentialStream)

- `#define chSequentialStreamWrite(ip, bp, n) ((ip)->vmt->write(ip, bp, n))`
Sequential Stream write.
- `#define chSequentialStreamRead(ip, bp, n) ((ip)->vmt->read(ip, bp, n))`
Sequential Stream read.
- `#define chSequentialStreamPut(ip, b) ((ip)->vmt->put(ip, b))`
Sequential Stream blocking byte write.
- `#define chSequentialStreamGet(ip) ((ip)->vmt->get(ip))`
Sequential Stream blocking byte read.

Data Structures

- struct `BaseSequentialStreamVMT`
BaseSequentialStream virtual methods table.
- struct `BaseSequentialStream`
Base stream class.

9.25.2 Macro Definition Documentation

9.25.2.1 #define _base_sequential_stream_methods

Value:

```
/* Stream write buffer method.*/
size_t (*write)(void *instance, const uint8_t *bp, size_t n);
/* Stream read buffer method.*/
size_t (*read)(void *instance, uint8_t *bp, size_t n);
/* Channel put method, blocking.*/
msg_t (*put)(void *instance, uint8_t b);
/* Channel get method, blocking.*/
msg_t (*get)(void *instance);
```



`BaseSequentialStream` specific methods.

9.25.2.2 #define _base_sequential_stream_data

`BaseSequentialStream` specific data.

Note

It is empty because `BaseSequentialStream` is only an interface without implementation.

9.25.2.3 #define chSequentialStreamWrite(ip, bp, n) ((ip)->vmt->write(ip, bp, n))

Sequential Stream write.

The function writes data from a buffer to a stream.

Parameters

in	<i>ip</i>	pointer to a <code>BaseSequentialStream</code> or derived class
in	<i>bp</i>	pointer to the data buffer
in	<i>n</i>	the maximum amount of data to be transferred

Returns

The number of bytes transferred. The return value can be less than the specified number of bytes if an end-of-file condition has been met.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

9.25.2.4 #define chSequentialStreamRead(ip, bp, n) ((ip)->vmt->read(ip, bp, n))

Sequential Stream read.

The function reads data from a stream into a buffer.

Parameters

in	<i>ip</i>	pointer to a <code>BaseSequentialStream</code> or derived class
out	<i>bp</i>	pointer to the data buffer
in	<i>n</i>	the maximum amount of data to be transferred

Returns

The number of bytes transferred. The return value can be less than the specified number of bytes if an end-of-file condition has been met.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

9.25.2.5 #define chSequentialStreamPut(ip, b) ((ip)->vmt->put(ip, b))

Sequential Stream blocking byte write.

This function writes a byte value to a channel. If the channel is not ready to accept data then the calling thread is suspended.

Parameters

in	<i>ip</i>	pointer to a BaseSequentialStream or derived class
in	<i>b</i>	the byte value to be written to the channel

Returns

The operation status.

Return values

<i>Q_OK</i>	if the operation succeeded.
<i>Q_RESET</i>	if an end-of-file condition has been met.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

9.25.2.6 #define chSequentialStreamGet(*ip*) ((ip)->vmt->get(ip))

Sequential Stream blocking byte read.

This function reads a byte value from a channel. If the data is not available then the calling thread is suspended.

Parameters

in	<i>ip</i>	pointer to a BaseSequentialStream or derived class
----	-----------	--

Returns

A byte value from the queue.

Return values

<i>Q_RESET</i>	if an end-of-file condition has been met.
----------------	---

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

9.26 Registry

9.26.1 Detailed Description

Threads Registry related APIs and services.

Operation mode

The Threads Registry is a double linked list that holds all the active threads in the system.
Operations defined for the registry:

- **First**, returns the first, in creation order, active thread in the system.
- **Next**, returns the next, in creation order, active thread in the system.

The registry is meant to be mainly a debug feature, for example, using the registry a debugger can enumerate the active threads in any given moment or the shell can print the active threads and their state.
Another possible use is for centralized threads memory management, terminating threads can pulse an event source and an event handler can perform a scansion of the registry in order to recover the memory.

Precondition

In order to use the threads registry the `CH_CFG_USE_REGISTRY` option must be enabled in [chconf.h](#).

Macros

- `#define REG_REMOVE(tp)`
Removes a thread from the registry list.
- `#define REG_INSERT(tp)`
Adds a thread to the registry list.

Data Structures

- struct `chdebug_t`
ChibiOS/RT memory signature record.

Functions

- `thread_t * chRegFirstThread (void)`
Returns the first thread in the system.
- `thread_t * chRegNextThread (thread_t *tp)`
Returns the thread next to the specified one.
- static void `chRegSetThreadName (const char *name)`
Sets the current thread name.
- static const char * `chRegGetThreadNameX (thread_t *tp)`
Returns the name of the specified thread.
- static void `chRegSetThreadNameX (thread_t *tp, const char *name)`
Changes the name of the specified thread.

9.26.2 Macro Definition Documentation

9.26.2.1 #define REG_REMOVE(*tp*)

Value:

```
{
    (tp)->p_older->p_newer = (tp)->p_newer;
    (tp)->p_newer->p_older = (tp)->p_older;
}
```

Removes a thread from the registry list.

Note

This macro is not meant for use in application code.

Parameters

in	<i>tp</i>	thread to remove from the registry
----	-----------	------------------------------------

9.26.2.2 #define REG_INSERT(*tp*)

Value:

```
{
    (tp)->p_newer = (thread_t *)&ch.rlist;
    (tp)->p_older = ch.rlist.r_older;
    (tp)->p_older->p_newer = (tp);
    ch.rlist.r_older = (tp);
}
```

Adds a thread to the registry list.

Note

This macro is not meant for use in application code.

Parameters

in	<i>tp</i>	thread to add to the registry
----	-----------	-------------------------------

9.26.3 Function Documentation

9.26.3.1 *thread_t* * chRegFirstThread(void)

Returns the first thread in the system.

Returns the most ancient thread in the system, usually this is the main thread unless it terminated. A reference is added to the returned thread in order to make sure its status is not lost.

Note

This function cannot return NULL because there is always at least one thread in the system.

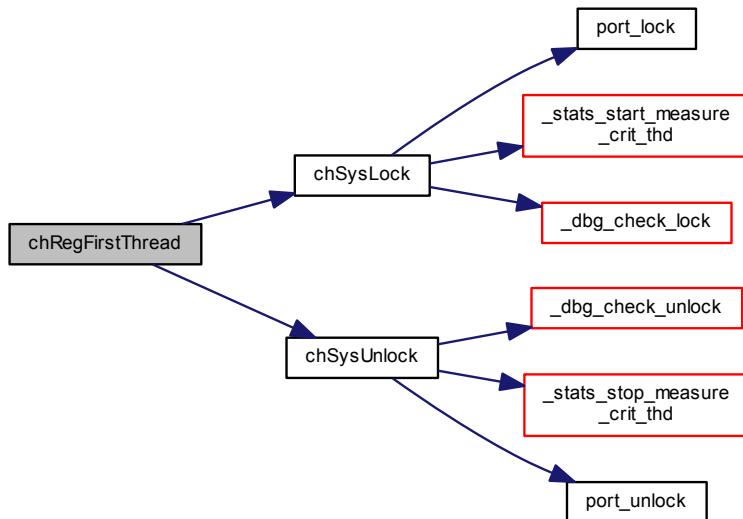
Returns

A reference to the most ancient thread.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

**9.26.3.2 `thread_t * chRegNextThread(thread_t * tp)`**

Returns the thread next to the specified one.

The reference counter of the specified thread is decremented and the reference counter of the returned thread is incremented.

Parameters

in	<code>tp</code>	pointer to the thread
----	-----------------	-----------------------

Returns

A reference to the next thread.

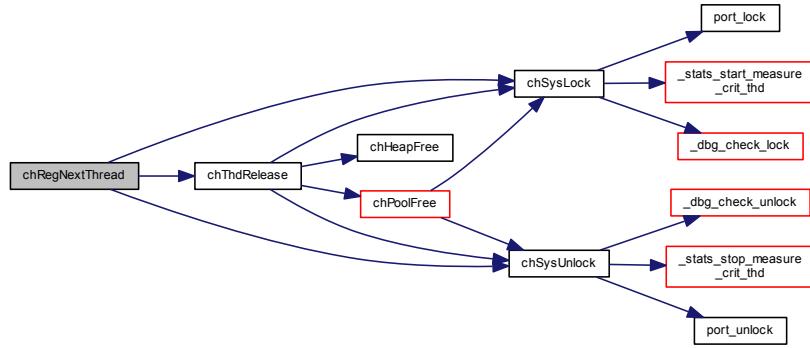
Return values

<code>NULL</code>	if there is no next thread.
-------------------	-----------------------------

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



9.26.3.3 static void chRegSetThreadName(const char * name) [inline], [static]

Sets the current thread name.

Precondition

This function only stores the pointer to the name if the option `CH_CFG_USE_REGISTRY` is enabled else no action is performed.

Parameters

in	<i>name</i>	thread name as a zero terminated string
----	-------------	---

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

9.26.3.4 static const char* chRegGetThreadNameX(thread_t * tp) [inline], [static]

Returns the name of the specified thread.

Precondition

This function only returns the pointer to the name if the option `CH_CFG_USE_REGISTRY` is enabled else `NULL` is returned.

Parameters

in	<i>tp</i>	pointer to the thread
----	-----------	-----------------------

Returns

Thread name as a zero terminated string.

Return values

<code>NULL</code>	if the thread name has not been set.
-------------------	--------------------------------------

9.26.3.5 static void chRegSetThreadNameX(`thread_t` * *tp*, `const char` * *name*) [inline], [static]

Changes the name of the specified thread.

Precondition

This function only stores the pointer to the name if the option CH_CFG_USE_REGISTRY is enabled else no action is performed.

Parameters

in	<i>tp</i>	pointer to the thread
in	<i>name</i>	thread name as a zero terminated string

Function Class:

This is an **X-Class** API, this function can be invoked from any context.

9.27 Debug

9.27.1 Detailed Description

Debug APIs and services:

- Runtime system state and call protocol check. The following panic messages can be generated:
 - SV#1, misplaced `chSysDisable()`.
 - * Called from an ISR.
 - * Called from a critical zone.
 - SV#2, misplaced `chSysSuspend()`
 - * Called from an ISR.
 - * Called from a critical zone.
 - SV#3, misplaced `chSysEnable()`.
 - * Called from an ISR.
 - * Called from a critical zone.
 - SV#4, misplaced `chSysLock()`.
 - * Called from an ISR.
 - * Called from a critical zone.
 - SV#5, misplaced `chSysUnlock()`.
 - * Called from an ISR.
 - * Not called from a critical zone.
 - SV#6, misplaced `chSysLockFromISR()`.
 - * Not called from an ISR.
 - * Called from a critical zone.
 - SV#7, misplaced `chSysUnlockFromISR()`.
 - * Not called from an ISR.
 - * Not called from a critical zone.
 - SV#8, misplaced `CH_IRQ_PROLOGUE()`.
 - * Not called at ISR begin.
 - * Called from a critical zone.
 - SV#9, misplaced `CH_IRQ_EPILOGUE()`.
 - * `CH_IRQ_PROLOGUE()` missing.
 - * Not called at ISR end.
 - * Called from a critical zone.
 - SV#10, misplaced I-class function.
 - * I-class function not called from within a critical zone.
 - SV#11, misplaced S-class function.
 - * S-class function not called from within a critical zone.
 - * Called from an ISR.
- Trace buffer.
- Parameters check.
- Kernel assertions.
- Kernel panics.

Note

Stack checks are not implemented in this module but in the port layer in an architecture-dependent way.

Debug related settings

- `#define CH_DBG_TRACE_BUFFER_SIZE 64`
Trace buffer entries.
- `#define CH_DBG_STACK_FILL_VALUE 0x55`
Fill value for thread stack area in debug mode.
- `#define CH_DBG_THREAD_FILL_VALUE 0xFF`
Fill value for thread area in debug mode.

Macro Functions

- `#define chDbgCheck(c)`
Function parameters check.
- `#define chDbgAssert(c, r)`
Condition assertion.

Data Structures

- struct `ch_swc_event_t`
Trace buffer record.
- struct `ch_trace_buffer_t`
Trace buffer header.

Functions

- `void _dbg_check_disable (void)`
Guard code for `chSysDisable ()`.
- `void _dbg_check_suspend (void)`
Guard code for `chSysSuspend ()`.
- `void _dbg_check_enable (void)`
Guard code for `chSysEnable ()`.
- `void _dbg_check_lock (void)`
Guard code for `chSysLock ()`.
- `void _dbg_check_unlock (void)`
Guard code for `chSysUnlock ()`.
- `void _dbg_check_lock_from_isr (void)`
Guard code for `chSysLockFromIsr ()`.
- `void _dbg_check_unlock_from_isr (void)`
Guard code for `chSysUnlockFromIsr ()`.
- `void _dbg_check_enter_isr (void)`
Guard code for `CH IRQ PROLOGUE ()`.
- `void _dbg_check_leave_isr (void)`
Guard code for `CH IRQ EPILOGUE ()`.
- `void chDbgCheckClassI (void)`
I-class functions context check.
- `void chDbgCheckClassS (void)`
S-class functions context check.
- `void _dbg_trace_init (void)`
Trace circular buffer subsystem initialization.
- `void _dbg_trace (thread_t *otp)`
Inserts in the circular debug trace buffer a context switch record.

9.27.2 Macro Definition Documentation

9.27.2.1 #define CH_DBG_TRACE_BUFFER_SIZE 64

Trace buffer entries.

9.27.2.2 #define CH_DBG_STACK_FILL_VALUE 0x55

Fill value for thread stack area in debug mode.

9.27.2.3 #define CH_DBG_THREAD_FILL_VALUE 0xFF

Fill value for thread area in debug mode.

Note

The chosen default value is 0xFF in order to make evident which thread fields were not initialized when inspecting the memory with a debugger. A uninitialized field is not an error in itself but it better to know it.

9.27.2.4 #define chDbgCheck(c)

Value:

```
do {
    /*lint -save -e506 -e774 [2.1, 14.3] Can be a constant by design.*/
    if (CH_DBG_ENABLE_CHECKS != FALSE) {
        if (!(c)) {
            /*lint -restore*/
            chSysHalt(__func__);
        }
    }
} while (false)
```

Function parameters check.

If the condition check fails then the kernel panics and halts.

Note

The condition is tested only if the CH_DBG_ENABLE_CHECKS switch is specified in `chconf.h` else the macro does nothing.

Parameters

in	c	the condition to be verified to be true
----	---	---

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

9.27.2.5 #define chDbgAssert(c, r)

Value:

```
do {
    /*lint -save -e506 -e774 [2.1, 14.3] Can be a constant by design.*/
    if (CH_DBG_ENABLE_ASSERTS != FALSE) {
        if (!(c)) {
            /*lint -restore*/
            chSysHalt(__func__);
        }
    }
} while (false)
```

Condition assertion.

If the condition check fails then the kernel panics with a message and halts.

Note

The condition is tested only if the CH_DBG_ENABLE_ASSERTS switch is specified in [chconf.h](#) else the macro does nothing.

The remark string is not currently used except for putting a comment in the code about the assertion.

Parameters

in	<i>c</i>	the condition to be verified to be true
in	<i>r</i>	a remark string

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

9.27.3 Function Documentation

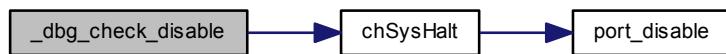
9.27.3.1 void _dbg_check_disable(void)

Guard code for [chSysDisable\(\)](#).

Function Class:

Not an API, this function is for internal use only.

Here is the call graph for this function:



9.27.3.2 void _dbg_check_suspend(void)

Guard code for [chSysSuspend\(\)](#).

Function Class:

Not an API, this function is for internal use only.

Here is the call graph for this function:



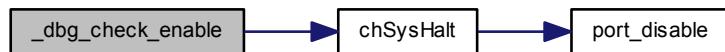
9.27.3.3 void _dbg_check_enable (void)

Guard code for [chSysEnable\(\)](#).

Function Class:

Not an API, this function is for internal use only.

Here is the call graph for this function:

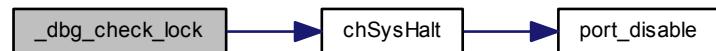
**9.27.3.4 void _dbg_check_lock (void)**

Guard code for [chSysLock\(\)](#).

Function Class:

Not an API, this function is for internal use only.

Here is the call graph for this function:

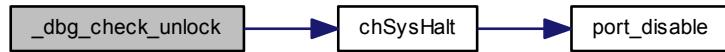
**9.27.3.5 void _dbg_check_unlock (void)**

Guard code for [chSysUnlock\(\)](#).

Function Class:

Not an API, this function is for internal use only.

Here is the call graph for this function:



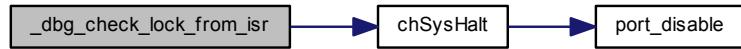
9.27.3.6 void _dbg_check_lock_from_isr(void)

Guard code for `chSysLockFromIsr()`.

Function Class:

Not an API, this function is for internal use only.

Here is the call graph for this function:



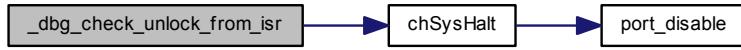
9.27.3.7 void _dbg_check_unlock_from_isr(void)

Guard code for `chSysUnlockFromIsr()`.

Function Class:

Not an API, this function is for internal use only.

Here is the call graph for this function:



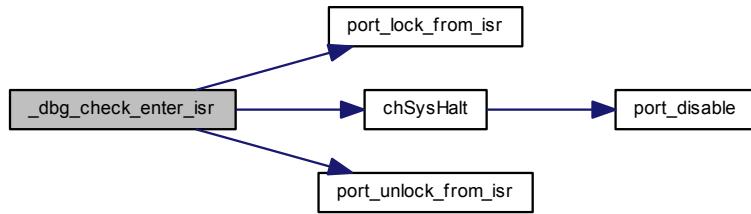
9.27.3.8 void _dbg_check_enter_isr(void)

Guard code for `CH_IRQ_PROLOGUE()`.

Function Class:

Not an API, this function is for internal use only.

Here is the call graph for this function:

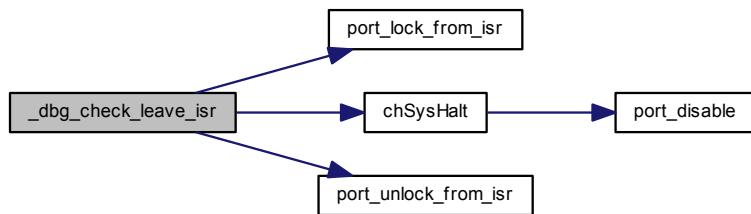
**9.27.3.9 void _dbg_check_leave_isr(void)**

Guard code for [CH_IRQ_EPILOGUE\(\)](#).

Function Class:

Not an API, this function is for internal use only.

Here is the call graph for this function:

**9.27.3.10 void chDbgCheckClassI(void)**

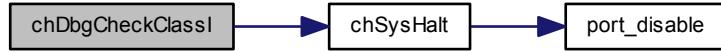
I-class functions context check.

Verifies that the system is in an appropriate state for invoking an I-class API function. A panic is generated if the state is not compatible.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



9.27.3.11 void chDbgCheckClassS (void)

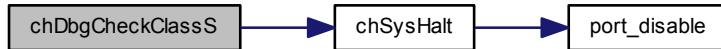
S-class functions context check.

Verifies that the system is in an appropriate state for invoking an S-class API function. A panic is generated if the state is not compatible.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



9.27.3.12 void _dbg_trace_init (void)

Trace circular buffer subsystem initialization.

Note

Internal use only.

9.27.3.13 void _dbg_trace (thread_t * otp)

Inserts in the circular debug trace buffer a context switch record.

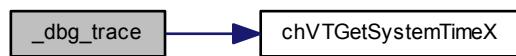
Parameters

in	otp	the thread being switched out
----	-----	-------------------------------

Function Class:

Not an API, this function is for internal use only.

Here is the call graph for this function:



9.28 Time Measurement

9.28.1 Detailed Description

Time Measurement APIs and services.

Data Structures

- struct `tm_calibration_t`
Type of a time measurement calibration data.
- struct `time_measurement_t`
Type of a Time Measurement object.

Functions

- void `_tm_init (void)`
Initializes the time measurement unit.
- void `chTMObjectInit (time_measurement_t *tmp)`
Initializes a TimeMeasurement object.
- `NOINLINE void chTMStartMeasurementX (time_measurement_t *tmp)`
Starts a measurement.
- `NOINLINE void chTMStopMeasurementX (time_measurement_t *tmp)`
Stops a measurement.
- `NOINLINE void chTMChainMeasurementToX (time_measurement_t *tmp1, time_measurement_t *tmp2)`
Stops a measurement and chains to the next one using the same time stamp.

9.28.2 Function Documentation

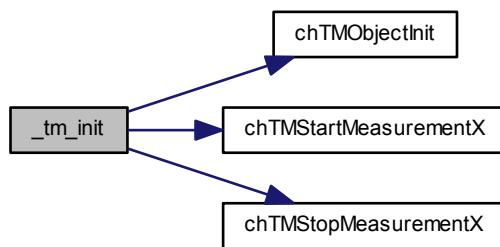
9.28.2.1 void `_tm_init (void)`

Initializes the time measurement unit.

Function Class:

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

Here is the call graph for this function:



9.28.2.2 void chTMObjectInit(time_measurement_t * *tmp*)

Initializes a TimeMeasurement object.

Parameters

out	<i>tmp</i>	pointer to a TimeMeasurement structure
-----	------------	--

Function Class:

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

9.28.2.3 NOINLINE void chTMStartMeasurementX (time_measurement_t * *tmp*)

Starts a measurement.

Precondition

The [time_measurement_t](#) structure must be initialized.

Parameters

in, out	<i>tmp</i>	pointer to a TimeMeasurement structure
---------	------------	--

Function Class:

This is an **X-Class** API, this function can be invoked from any context.

9.28.2.4 NOINLINE void chTMStopMeasurementX (time_measurement_t * *tmp*)

Stops a measurement.

Precondition

The [time_measurement_t](#) structure must be initialized.

Parameters

in, out	<i>tmp</i>	pointer to a time_measurement_t structure
---------	------------	---

Function Class:

This is an **X-Class** API, this function can be invoked from any context.

9.28.2.5 NOINLINE void chTMChainMeasurementToX (time_measurement_t * *tmp1*, time_measurement_t * *tmp2*)

Stops a measurement and chains to the next one using the same time stamp.

Parameters

in, out	<i>tmp1</i>	pointer to the time_measurement_t structure to be stopped
in, out	<i>tmp2</i>	pointer to the time_measurement_t structure to be started

Function Class:

This is an **X-Class** API, this function can be invoked from any context.

9.29 Statistics

9.29.1 Detailed Description

Statistics services.

Data Structures

- struct `kernel_stats_t`
Type of a kernel statistics structure.

Functions

- void `_stats_init (void)`
Initializes the statistics module.
- void `_stats_increase_irq (void)`
Increases the IRQ counter.
- void `_stats_ctxswc (thread_t *ntp, thread_t *otp)`
Updates context switch related statistics.
- void `_stats_start_measure_crit_thd (void)`
Starts the measurement of a thread critical zone.
- void `_stats_stop_measure_crit_thd (void)`
Stops the measurement of a thread critical zone.
- void `_stats_start_measure_crit_isr (void)`
Starts the measurement of an ISR critical zone.
- void `_stats_stop_measure_crit_isr (void)`
Stops the measurement of an ISR critical zone.

9.29.2 Function Documentation

9.29.2.1 void `_stats_init (void)`

Initializes the statistics module.

Function Class:

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

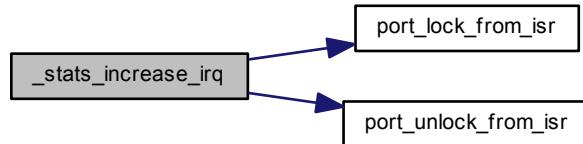
Here is the call graph for this function:



9.29.2.2 void _stats_increase_irq(void)

Increases the IRQ counter.

Here is the call graph for this function:



9.29.2.3 void _stats_ctxswc(thread_t *ntp, thread_t *otp)

Updates context switch related statistics.

Parameters

in	<i>ntp</i>	the thread to be switched in
in	<i>otp</i>	the thread to be switched out

Here is the call graph for this function:



9.29.2.4 void _stats_start_measure_crit_thd(void)

Starts the measurement of a thread critical zone.

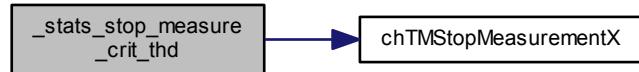
Here is the call graph for this function:



9.29.2.5 void _stats_stop_measure_crit_thd (void)

Stops the measurement of a thread critical zone.

Here is the call graph for this function:

**9.29.2.6 void _stats_start_measure_crit_isr (void)**

Starts the measurement of an ISR critical zone.

Here is the call graph for this function:

**9.29.2.7 void _stats_stop_measure_crit_isr (void)**

Stops the measurement of an ISR critical zone.

Here is the call graph for this function:



9.30 Port Layer

9.30.1 Detailed Description

Non portable code templates.

Macros

- `#define PORT_IDLE_THREAD_STACK_SIZE 32`
Stack size for the system idle thread.
- `#define PORT_INT_REQUIRED_STACK 256`
Per-thread stack overhead for interrupts servicing.
- `#define PORT_USE_ALT_TIMER FALSE`
Enables an alternative timer implementation.
- `#define PORT_SETUP_CONTEXT(tp, workspace, wsize, pf, arg)`
Platform dependent part of the `chThdCreateI()` API.
- `#define PORT_WA_SIZE(n)`
Computes the thread working area global size.
- `#define PORT_IRQ_IS_VALID_PRIORITY(n) false`
Priority level verification macro.
- `#define PORT_IRQ_IS_VALID_KERNEL_PRIORITY(n) false`
Priority level verification macro.
- `#define PORT_IRQ_PROLOGUE()`
IRQ prologue code.
- `#define PORT_IRQ_EPILOGUE()`
IRQ epilogue code.
- `#define PORT_IRQ_HANDLER(id) void id(void)`
IRQ handler function declaration.
- `#define PORT_FAST_IRQ_HANDLER(id) void id(void)`
Fast IRQ handler function declaration.
- `#define port_switch(ntp, otp) _port_switch(ntp, otp)`
Performs a context switch between two threads.

Architecture and Compiler

- `#define PORT_ARCHITECTURE_XXX`
Macro defining an XXX architecture.
- `#define PORT_ARCHITECTURE_XXX_YYY`
Macro defining the specific XXX architecture.
- `#define PORT_ARCHITECTURE_NAME "XXX Architecture"`
Name of the implemented architecture.
- `#define PORT_COMPILER_NAME "GCC" __VERSION__`
Compiler name and version.
- `#define PORT_SUPPORTS_RT FALSE`
This port supports a realtime counter.
- `#define PORT_INFO "no info"`
Port-specific information string.

Typedefs

- `typedef uint64_t stkalign_t`
Type of stack and memory alignment enforcement.

Data Structures

- `struct port_extctx`
Interrupt saved context.
- `struct port_intctx`
System saved context.
- `struct context`
Platform dependent part of the thread_t structure.

Functions

- `void _port_init (void)`
Port-related initialization code.
- `void _port_switch (thread_t *ntp, thread_t *otp)`
Performs a context switch between two threads.
- `static syssts_t port_get_irq_status (void)`
Returns a word encoding the current interrupts status.
- `static bool port_irq_enabled (syssts_t sts)`
Checks the interrupt status.
- `static bool port_is_isr_context (void)`
Determines the current execution context.
- `static void port_lock (void)`
Kernel-lock action.
- `static void port_unlock (void)`
Kernel-unlock action.
- `static void port_lock_from_isr (void)`
Kernel-lock action from an interrupt handler.
- `static void port_unlock_from_isr (void)`
Kernel-unlock action from an interrupt handler.
- `static void port_disable (void)`
Disables all the interrupt sources.
- `static void port_suspend (void)`
Disables the interrupt sources below kernel-level priority.
- `static void port_enable (void)`
Enables all the interrupt sources.
- `static void port_wait_for_interrupt (void)`
Enters an architecture-dependent IRQ-waiting mode.
- `static rtcnt_t port_rt_get_counter_value (void)`
Returns the current value of the realtime counter.

9.30.2 Macro Definition Documentation

9.30.2.1 #define PORT_ARCHITECTURE_XXX

Macro defining an XXX architecture.

9.30.2.2 #define PORT_ARCHITECTURE_XXX_YYY

Macro defining the specific XXX architecture.

9.30.2.3 #define PORT_ARCHITECTURE_NAME "XXX Architecture"

Name of the implemented architecture.

9.30.2.4 #define PORT_COMPILER_NAME "GCC" __VERSION__

Compiler name and version.

9.30.2.5 #define PORT_SUPPORTS_RT FALSE

This port supports a realtime counter.

9.30.2.6 #define PORT_INFO "no info"

Port-specific information string.

9.30.2.7 #define PORT_IDLE_THREAD_STACK_SIZE 32

Stack size for the system idle thread.

This size depends on the idle thread implementation, usually the idle thread should take no more space than those reserved by `PORT_INT_REQUIRED_STACK`.

9.30.2.8 #define PORT_INT_REQUIRED_STACK 256

Per-thread stack overhead for interrupts servicing.

This constant is used in the calculation of the correct working area size.

9.30.2.9 #define PORT_USE_ALT_TIMER FALSE

Enables an alternative timer implementation.

Usually the port uses a timer interface defined in the file `chcore_timer.h`, if this option is enabled then the file `chcore_timer_alt.h` is included instead.

9.30.2.10 #define PORT_SETUP_CONTEXT(tp, workspace, wsize, pf, arg)

Value:

```
{           \
```

Platform dependent part of the `chThdCreateI()` API.

This code usually setup the context switching frame represented by an `port_intctx` structure.

9.30.2.11 #define PORT_WA_SIZE(*n*)**Value:**

```
(sizeof(struct port_intctx) + \
    sizeof(struct port_extctx) + \
    ((size_t)(n)) + ((size_t)(PORT_INT_REQUIRED_STACK))) \
```

Computes the thread working area global size.

Note

There is no need to perform alignments in this macro.

9.30.2.12 #define PORT_IRQ_IS_VALID_PRIORITY(*n*) false

Priority level verification macro.

9.30.2.13 #define PORT_IRQ_IS_VALID_KERNEL_PRIORITY(*n*) false

Priority level verification macro.

9.30.2.14 #define PORT_IRQ_PROLOGUE()

IRQ prologue code.

This macro must be inserted at the start of all IRQ handlers enabled to invoke system APIs.

9.30.2.15 #define PORT_IRQ_EPILOGUE()

IRQ epilogue code.

This macro must be inserted at the end of all IRQ handlers enabled to invoke system APIs.

9.30.2.16 #define PORT_IRQ_HANDLER(*id*) void id(void)

IRQ handler function declaration.

Note

id can be a function name or a vector number depending on the port implementation.

9.30.2.17 #define PORT_FAST_IRQ_HANDLER(*id*) void id(void)

Fast IRQ handler function declaration.

Note

id can be a function name or a vector number depending on the port implementation.

9.30.2.18 #define port_switch(*ntp*, *otp*) _port_switch(*ntp*, *otp*)

Performs a context switch between two threads.

This is the most critical code in any port, this function is responsible for the context switch between 2 threads.

Note

The implementation of this code affects **directly** the context switch performance so optimize here as much as you can.

Parameters

in	<i>ntp</i>	the thread to be switched in
in	<i>otp</i>	the thread to be switched out

9.30.3 Typedef Documentation

9.30.3.1 typedef uint64_t stkalign_t

Type of stack and memory alignment enforcement.

Note

In this architecture the stack alignment is enforced to 64 bits.

9.30.4 Function Documentation

9.30.4.1 void _port_init(void)

Port-related initialization code.

Note

This function is usually empty.

9.30.4.2 void _port_switch(thread_t * *ntp*, thread_t * *otp*)

Performs a context switch between two threads.

This is the most critical code in any port, this function is responsible for the context switch between 2 threads.

Note

The implementation of this code affects **directly** the context switch performance so optimize here as much as you can.

Parameters

in	<i>ntp</i>	the thread to be switched in
in	<i>otp</i>	the thread to be switched out

9.30.4.3 static syssts_t port_get_irq_status(void) [inline], [static]

Returns a word encoding the current interrupts status.

Returns

The interrupts status.

9.30.4.4 static bool port_irq_enabled (syssts_t sts) [inline], [static]

Checks the interrupt status.

Parameters

in	sts	the interrupt status word
----	-----	---------------------------

Returns

The interrupt status.

Return values

<i>false</i>	the word specified a disabled interrupts status.
<i>true</i>	the word specified an enabled interrupts status.

9.30.4.5 static bool port_is_isr_context (void) [inline], [static]

Determines the current execution context.

Returns

The execution context.

Return values

<i>false</i>	not running in ISR mode.
<i>true</i>	running in ISR mode.

9.30.4.6 static void port_lock (void) [inline], [static]

Kernel-lock action.

Usually this function just disables interrupts but may perform more actions.

9.30.4.7 static void port_unlock (void) [inline], [static]

Kernel-unlock action.

Usually this function just enables interrupts but may perform more actions.

9.30.4.8 static void port_lock_from_isr (void) [inline], [static]

Kernel-lock action from an interrupt handler.

This function is invoked before invoking I-class APIs from interrupt handlers. The implementation is architecture dependent, in its simplest form it is void.

9.30.4.9 static void port_unlock_from_isr (void) [inline], [static]

Kernel-unlock action from an interrupt handler.

This function is invoked after invoking I-class APIs from interrupt handlers. The implementation is architecture dependent, in its simplest form it is void.

9.30.4.10 static void port_disable(void) [inline], [static]

Disables all the interrupt sources.

Note

Of course non-maskable interrupt sources are not included.

9.30.4.11 static void port_suspend(void) [inline], [static]

Disables the interrupt sources below kernel-level priority.

Note

Interrupt sources above kernel level remains enabled.

9.30.4.12 static void port_enable(void) [inline], [static]

Enables all the interrupt sources.

9.30.4.13 static void port_wait_for_interrupt(void) [inline], [static]

Enters an architecture-dependent IRQ-waiting mode.

The function is meant to return when an interrupt becomes pending. The simplest implementation is an empty function or macro but this would not take advantage of architecture-specific power saving modes.

9.30.4.14 static rtcnt_t port_rt_get_counter_value(void) [inline], [static]

Returns the current value of the realtime counter.

Returns

The realtime counter value.

9.31 Test Runtime

9.31.1 Detailed Description

Runtime code for the test suite execution, this code is not part of the OS and should not be included in user applications.

Macros

- `#define DELAY_BETWEEN_TESTS 200`
Delay inserted between test cases.
- `#define TEST_NO_BENCHMARKS FALSE`
If TRUE then benchmarks are not included.
- `#define test_fail(point)`
Test failure enforcement.
- `#define test_assert(point, condition, msg)`
Test assertion.
- `#define test_assert_lock(point, condition, msg)`
Test assertion with lock.
- `#define test_assert_sequence(point, expected)`
Test sequence assertion.
- `#define test_assert_time_window(point, start, end)`
Test time window assertion.

Data Structures

- struct `testcase`
Structure representing a test case.

Functions

- `void test_printn (uint32_t n)`
Prints a decimal unsigned number.
- `void test_print (const char *msgp)`
Prints a line without final end-of-line.
- `void test_println (const char *msgp)`
Prints a line.
- `void test_emit_token (char token)`
Emits a token into the tokens buffer.
- `void test_terminate_threads (void)`
Sets a termination request in all the test-spawned threads.
- `void test_wait_threads (void)`
Waits for the completion of all the test-spawned threads.
- `systime_t test_wait_tick (void)`
Delays execution until next system time tick.
- `void test_start_timer (unsigned ms)`
Starts the test timer.
- `void TestThread (void *p)`
Test execution thread function.

Variables

- bool `test_timer_done`
Set to TRUE when the test timer reaches its deadline.

9.31.2 Macro Definition Documentation

9.31.2.1 #define DELAY_BETWEEN_TESTS 200

Delay inserted between test cases.

9.31.2.2 #define TEST_NO_BENCHMARKS FALSE

If TRUE then benchmarks are not included.

9.31.2.3 #define test_fail(point)

Value:

```
{
    _test_fail(point);
    return;
}
```

Test failure enforcement.

9.31.2.4 #define test_assert(point, condition, msg)

Value:

```
{
    if (_test_assert(point, condition))
        return;
}
```

Test assertion.

Parameters

in	<i>point</i>	numeric assertion identifier
in	<i>condition</i>	a boolean expression that must be verified to be true
in	<i>msg</i>	failure message

9.31.2.5 #define test_assert_lock(point, condition, msg)

Value:

```
{
    chSysLock();
    if (_test_assert(point, condition)) {
        chSysUnlock();
        return;
    }
    chSysUnlock();
}
```

Test assertion with lock.

Parameters

in	<i>point</i>	numeric assertion identifier
in	<i>condition</i>	a boolean expression that must be verified to be true
in	<i>msg</i>	failure message

9.31.2.6 #define test_assert_sequence(*point*, *expected*)**Value:**

```
{
    if (_test_assert_sequence(point, expected))
        return;
}
```

Test sequence assertion.

Parameters

in	<i>point</i>	numeric assertion identifier
in	<i>expected</i>	string to be matched with the tokens buffer

9.31.2.7 #define test_assert_time_window(*point*, *start*, *end*)**Value:**

```
{
    if (_test_assert_time_window(point, start, end))
        return;
}
```

Test time window assertion.

Parameters

in	<i>point</i>	numeric assertion identifier
in	<i>start</i>	initial time in the window (included)
in	<i>end</i>	final time in the window (not included)

9.31.3 Function Documentation**9.31.3.1 void test_printf(uint32_t *n*)**

Prints a decimal unsigned number.

Parameters

in	<i>n</i>	the number to be printed
----	----------	--------------------------

9.31.3.2 void test_print(const char * *msg*)

Prints a line without final end-of-line.

Parameters

in	<i>msgp</i>	the message
----	-------------	-------------

9.31.3.3 void test.println (const char * *msgp*)

Prints a line.

Parameters

in	<i>msgp</i>	the message
----	-------------	-------------

Here is the call graph for this function:

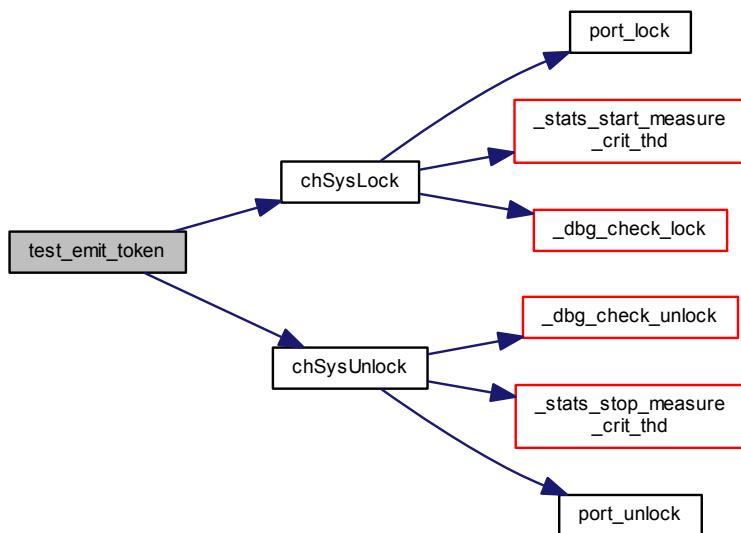
**9.31.3.4 void test.emit_token (char *token*)**

Emits a token into the tokens buffer.

Parameters

in	<i>token</i>	the token as a char
----	--------------	---------------------

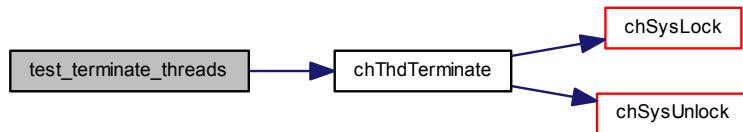
Here is the call graph for this function:



9.31.3.5 void test_terminate_threads (void)

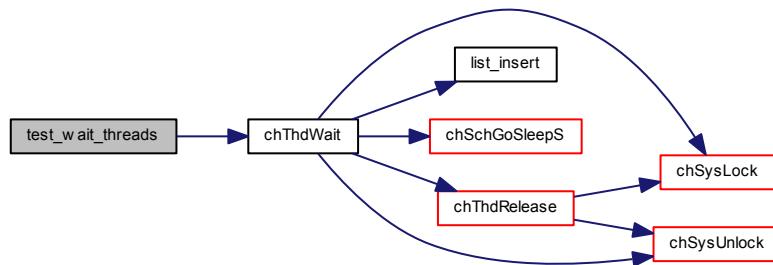
Sets a termination request in all the test-spawned threads.

Here is the call graph for this function:

**9.31.3.6 void test_wait_threads (void)**

Waits for the completion of all the test-spawned threads.

Here is the call graph for this function:

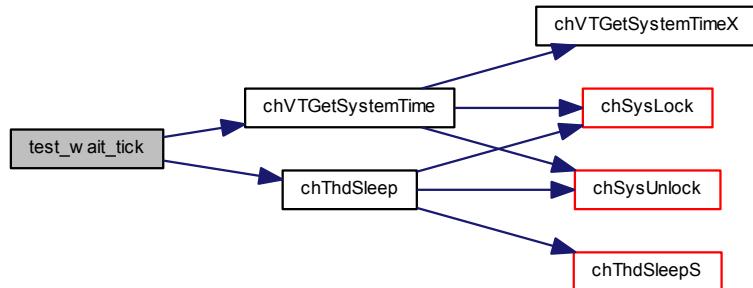
**9.31.3.7 systime_t test_wait_tick (void)**

Delays execution until next system time tick.

Returns

The system time.

Here is the call graph for this function:

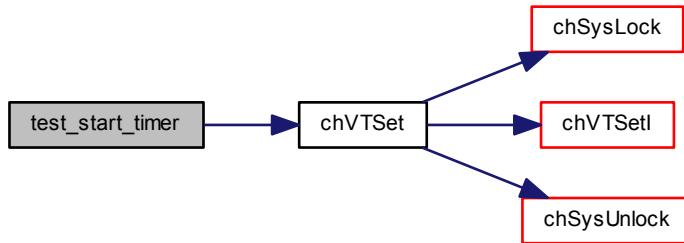
**9.31.3.8 void test_start_timer (unsigned ms)**

Starts the test timer.

Parameters

in	ms	time in milliseconds
----	----	----------------------

Here is the call graph for this function:

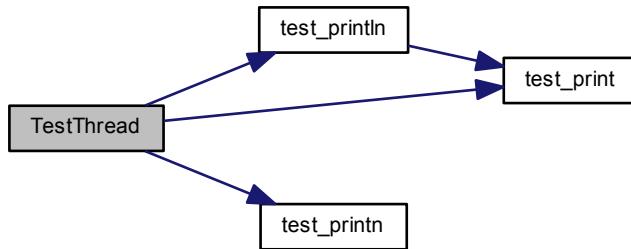
**9.31.3.9 void TestThread (void * p)**

Test execution thread function.

Parameters

in	<i>p</i>	pointer to a BaseChannel object for test output
----	----------	---

Here is the call graph for this function:



9.31.4 Variable Documentation

9.31.4.1 bool test_timer_done

Set to TRUE when the test timer reaches its deadline.

9.32 Customer

9.32.1 Detailed Description

Macros

- `#define CH_CUSTOMER_ID_STRING "Santa, North Pole"`
Customer readable identifier.
- `#define CH_CUSTOMER_ID_CODE "xxxx-yyyy"`
Customer code.

9.32.2 Macro Definition Documentation

9.32.2.1 `#define CH_CUSTOMER_ID_STRING "Santa, North Pole"`

Customer readable identifier.

9.32.2.2 `#define CH_CUSTOMER_ID_CODE "xxxx-yyyy"`

Customer code.

9.33 License

9.33.1 Detailed Description

Macros

- `#define CH_LICENSE CH_LICENSE_GPL`
Current license.
- `#define CH_LICENSE_TYPE_STRING "GNU General Public License 3 (GPL3)"`
License identification string.
- `#define CH_LICENSE_ID_STRING "N/A"`
Customer identification string.
- `#define CH_LICENSE_ID_CODE "N/A"`
Customer code.
- `#define CH_LICENSE_MODIFIABLE_CODE TRUE`
Code modifiability restrictions.
- `#define CH_LICENSE_FEATURES CH_FEATURES_FULL`
Code functionality restrictions.
- `#define CH_LICENSE_MAX_DEPLOY CH_DEPLOY_UNLIMITED`
Code deploy restrictions.

Allowed Features Levels

- `#define CH_FEATURES_BASIC 0`
- `#define CH_FEATURES_INTERMEDIATE 1`
- `#define CH_FEATURES_FULL 2`

Deployment Options

- `#define CH_DEPLOY_UNLIMITED -1`
- `#define CH_DEPLOY_NONE 0`

Licensing Options

- `#define CH_LICENSE_GPL 0`
- `#define CH_LICENSE_GPL_EXCEPTION 1`
- `#define CH_LICENSE_COMMERCIAL_FREE 2`
- `#define CH_LICENSE_COMMERCIAL_DEVELOPER 3`
- `#define CH_LICENSE_COMMERCIAL_FULL 4`
- `#define CH_LICENSE_PARTNER 5`

9.33.2 Macro Definition Documentation

9.33.2.1 `#define CH_LICENSE CH_LICENSE_GPL`

Current license.

Note

This setting is reserved to the copyright owner.

Changing this setting invalidates the license.

The license statement in the source headers is valid, applicable and binding regardless this setting.

9.33.2.2 #define CH_LICENSE_TYPE_STRING "GNU General Public License 3 (GPL3)"

License identification string.

This string identifies the license in a machine-readable format.

9.33.2.3 #define CH_LICENSE_ID_STRING "N/A"

Customer identification string.

This information is only available for registered commercial users.

9.33.2.4 #define CH_LICENSE_ID_CODE "N/A"

Customer code.

This information is only available for registered commercial users.

9.33.2.5 #define CH_LICENSE_MODIFIABLE_CODE TRUE

Code modifiability restrictions.

This setting defines if the source code is user-modifiable or not.

9.33.2.6 #define CH_LICENSE_FEATURES CH_FEATURES_FULL

Code functionality restrictions.

This setting defines which features are available under the current licensing scheme. The possible settings are:

- CH_FEATURES_FULL if all features are available.
- CH_FEATURES_INTERMEDIATE means that the following functionalities are disabled:
 - High Resolution mode.
 - Time Measurement.
 - Statistics.
- CH_FEATURES_BASIC means that the following functionalities are disabled:
 - High Resolution mode.
 - Time Measurement.
 - Statistics.
 - Tickless mode.
 - Recursive Mutexes.
 - Condition Variables.
 - Dynamic threading.

9.33.2.7 #define CH_LICENSE_MAX_DEPLOY CH_DEPLOY_UNLIMITED

Code deploy restrictions.

This is the per-core deploy limit allowed under the current license scheme.

Chapter 10

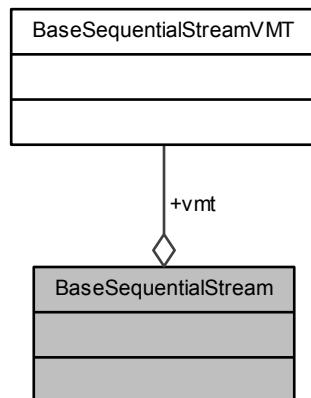
Data Structure Documentation

10.1 BaseSequentialStream Struct Reference

Base stream class.

```
#include <chstreams.h>
```

Collaboration diagram for BaseSequentialStream:



Data Fields

- const struct `BaseSequentialStreamVMT` * `vmt`

Virtual Methods Table.

10.1.1 Detailed Description

Base stream class.

This class represents a generic blocking unbuffered sequential data stream.

10.1.2 Field Documentation

10.1.2.1 const struct BaseSequentialStreamVMT* BaseSequentialStream::vmt

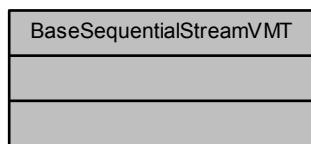
Virtual Methods Table.

10.2 BaseSequentialStreamVMT Struct Reference

[BaseSequentialStream](#) virtual methods table.

```
#include <chstreams.h>
```

Collaboration diagram for BaseSequentialStreamVMT:



10.2.1 Detailed Description

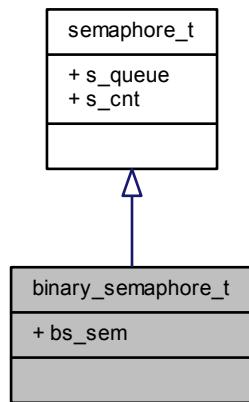
[BaseSequentialStream](#) virtual methods table.

10.3 binary_semaphore_t Struct Reference

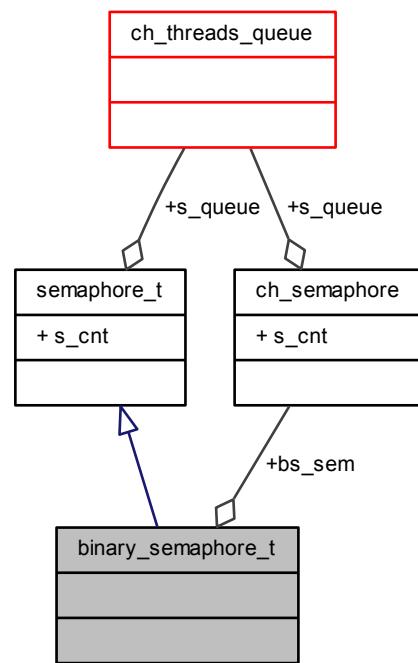
Binary semaphore type.

```
#include <chbsem.h>
```

Inheritance diagram for binary_semaphore_t:



Collaboration diagram for binary_semaphore_t:



Additional Inherited Members

10.3.1 Detailed Description

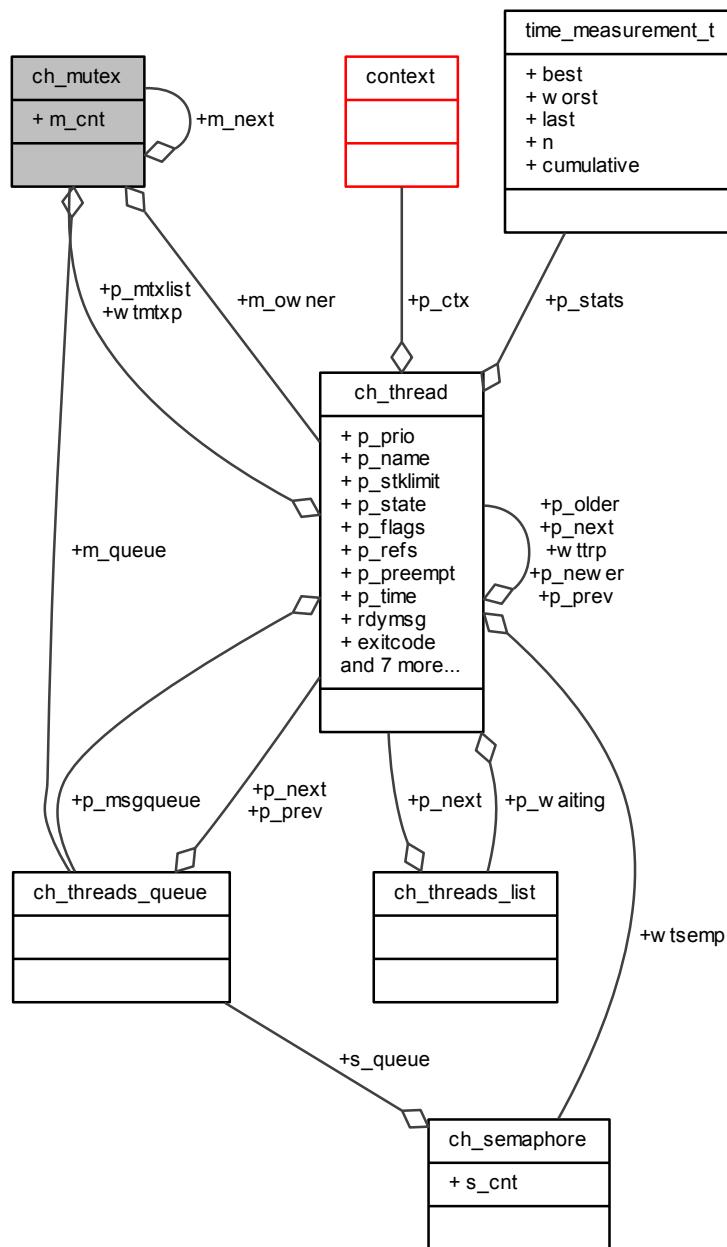
Binary semaphore type.

10.4 ch_mutex Struct Reference

Mutex structure.

```
#include <chmtx.h>
```

Collaboration diagram for ch_mutex:



Data Fields

- `threads_queue_t m_queue`
Queue of the threads sleeping on this mutex.
- `thread_t * m_owner`
Owner thread_t pointer or NULL.
- `mutex_t * m_next`
Next mutex_t into an owner-list or NULL.

- `cnt_t m_cnt`
Mutex recursion counter.

10.4.1 Detailed Description

Mutex structure.

10.4.2 Field Documentation

10.4.2.1 `threads_queue_t ch_mutex::m_queue`

Queue of the threads sleeping on this mutex.

10.4.2.2 `thread_t* ch_mutex::m_owner`

Owner `thread_t` pointer or `NULL`.

10.4.2.3 `mutex_t* ch_mutex::m_next`

Next `mutex_t` into an owner-list or `NULL`.

10.4.2.4 `cnt_t ch_mutex::m_cnt`

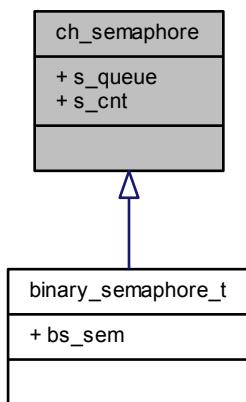
Mutex recursion counter.

10.5 ch_semaphore Struct Reference

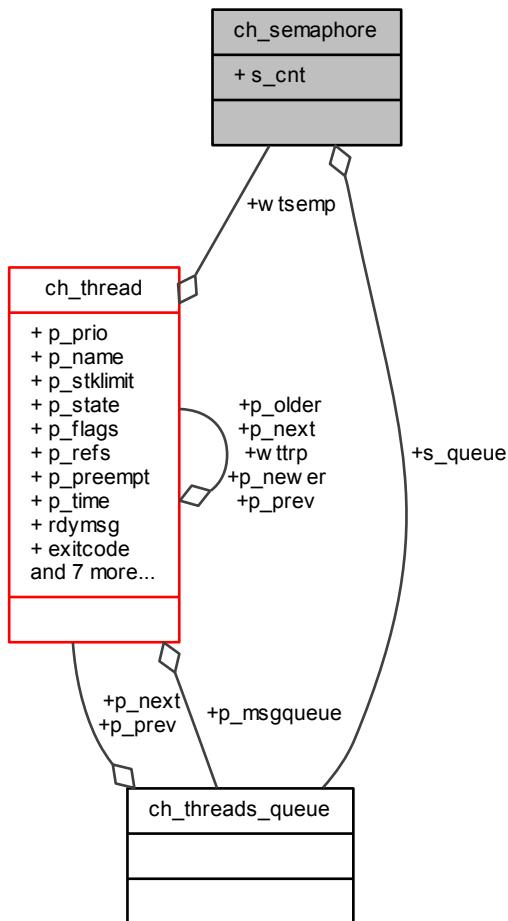
Semaphore structure.

```
#include <chsem.h>
```

Inheritance diagram for `ch_semaphore`:



Collaboration diagram for ch_semaphore:



Data Fields

- `threads_queue_t s_queue`
Queue of the threads sleeping on this semaphore.
- `cnt_t s_cnt`
The semaphore counter.

10.5.1 Detailed Description

Semaphore structure.

10.5.2 Field Documentation

10.5.2.1 `threads_queue_t ch_semaphore::s_queue`

Queue of the threads sleeping on this semaphore.

10.5.2.2 cnt_t ch_semaphore::s_cnt

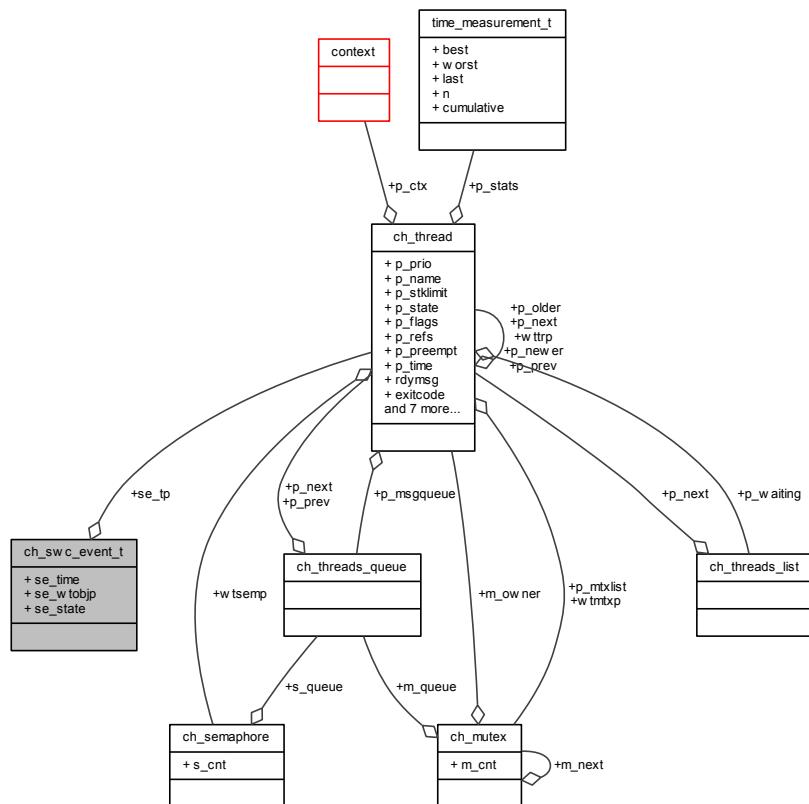
The semaphore counter.

10.6 ch_swc_event_t Struct Reference

Trace buffer record.

```
#include <chdebug.h>
```

Collaboration diagram for ch_swc_event_t:



Data Fields

- [systime_t se_time](#)
Time of the switch event.
- [thread_t * se_tp](#)
Switched in thread.
- [void * se_wtobjp](#)
Object where going to sleep.
- [uint8_t se_state](#)
Switched out thread state.

10.6.1 Detailed Description

Trace buffer record.

10.6.2 Field Documentation

10.6.2.1 systime_t ch_swc_event_t::se_time

Time of the switch event.

10.6.2.2 thread_t* ch_swc_event_t::se_tp

Switched in thread.

10.6.2.3 void* ch_swc_event_t::se_wtobjp

Object where going to sleep.

10.6.2.4 uint8_t ch_swc_event_t::se_state

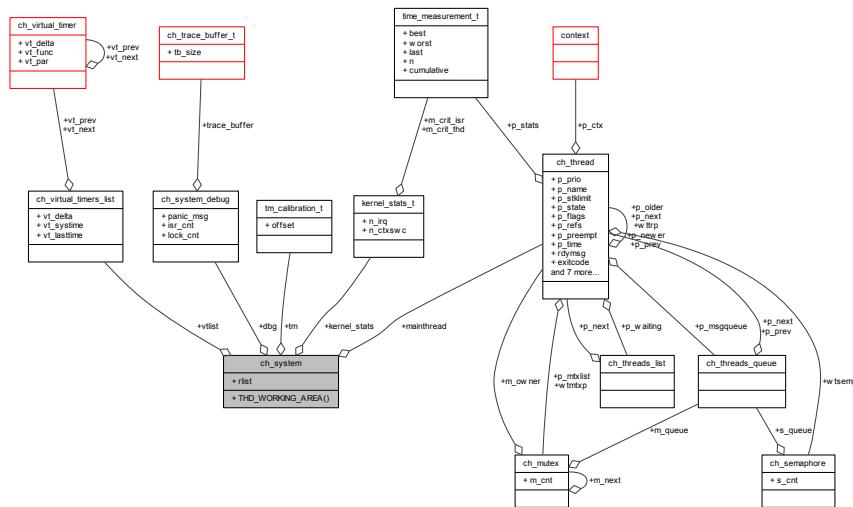
Switched out thread state.

10.7 ch_system Struct Reference

System data structure.

```
#include <chsched.h>
```

Collaboration diagram for ch_system:



Public Member Functions

- [THD_WORKING_AREA](#) (idle_thread_wa, PORT_IDLE_THREAD_STACK_SIZE)

Idle thread working area.

Data Fields

- [`ready_list_t rlist`](#)
Ready list header.
- [`virtual_timers_list_t vtlist`](#)
Virtual timers delta list header.
- [`system_debug_t dbg`](#)
System debug.
- [`thread_t mainthread`](#)
Main thread descriptor.
- [`tm_calibration_t tm`](#)
Time measurement calibration data.
- [`kernel_stats_t kernel_stats`](#)
Global kernel statistics.

10.7.1 Detailed Description

System data structure.

Note

This structure contain all the data areas used by the OS except stacks.

10.7.2 Member Function Documentation

10.7.2.1 `ch_system::THD_WORKING_AREA(idle_thread_wa , PORT_IDLE_THREAD_STACK_SIZE)`

Idle thread working area.

10.7.3 Field Documentation

10.7.3.1 `ready_list_t ch_system::rlist`

Ready list header.

10.7.3.2 `virtual_timers_list_t ch_system::vtlist`

Virtual timers delta list header.

10.7.3.3 `system_debug_t ch_system::dbg`

System debug.

10.7.3.4 `thread_t ch_system::mainthread`

Main thread descriptor.

10.7.3.5 tm_calibration_t ch_system::tm

Time measurement calibration data.

10.7.3.6 kernel_stats_t ch_system::kernel_stats

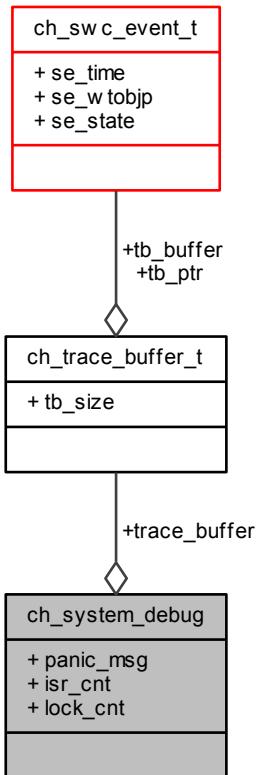
Global kernel statistics.

10.8 ch_system_debug Struct Reference

System debug data structure.

```
#include <chsched.h>
```

Collaboration diagram for ch_system_debug:



Data Fields

- const char *volatile panic_msg

Pointer to the panic message.

- cnt_t isr_cnt

ISR nesting level.

- `cnt_t lock_cnt`
Lock nesting level.
- `ch_trace_buffer_t trace_buffer`
Public trace buffer.

10.8.1 Detailed Description

System debug data structure.

10.8.2 Field Documentation

10.8.2.1 `const char* volatile ch_system_debug::panic_msg`

Pointer to the panic message.

This pointer is meant to be accessed through the debugger, it is written once and then the system is halted.

Note

Accesses to this pointer must never be optimized out so the field itself is declared volatile.

10.8.2.2 `cnt_t ch_system_debug::isr_cnt`

ISR nesting level.

10.8.2.3 `cnt_t ch_system_debug::lock_cnt`

Lock nesting level.

10.8.2.4 `ch_trace_buffer_t ch_system_debug::trace_buffer`

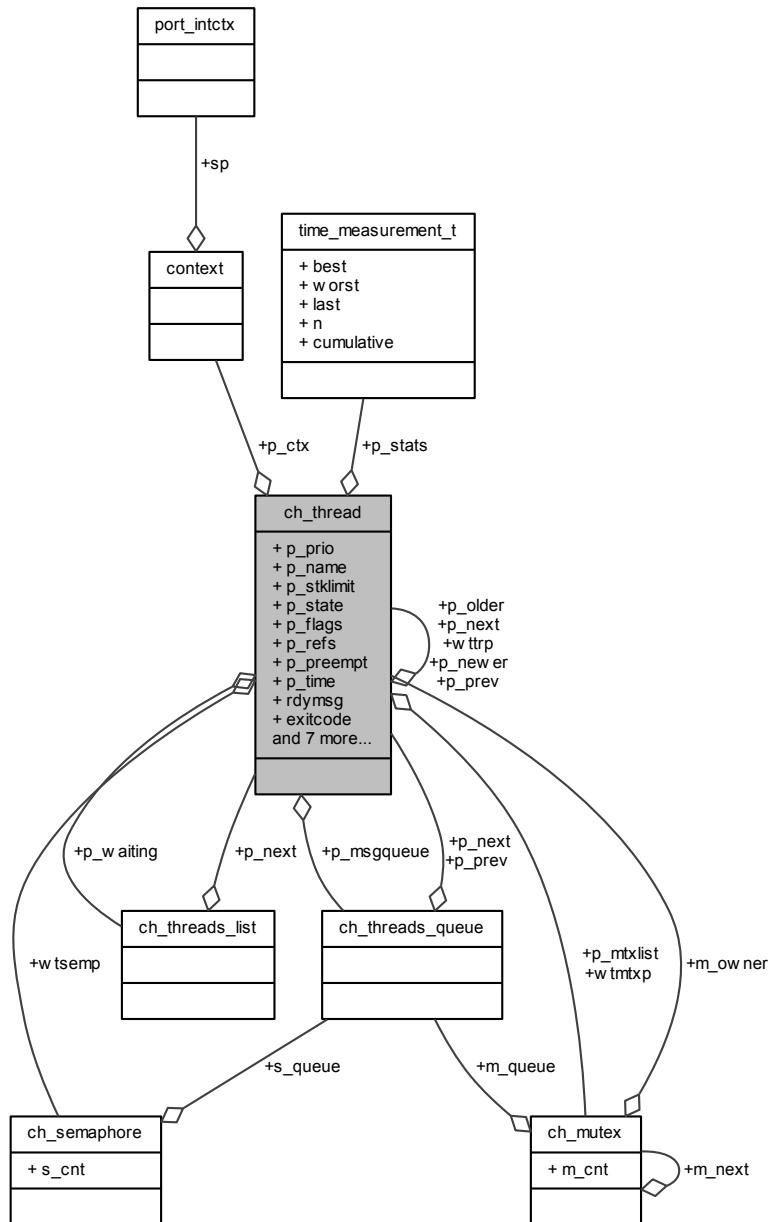
Public trace buffer.

10.9 `ch_thread` Struct Reference

Structure representing a thread.

```
#include <chsched.h>
```

Collaboration diagram for ch_thread:



Data Fields

- `thread_t * p_next`
Next in the list/queue.
- `thread_t * p_prev`
Previous in the queue.
- `tprio_t p_prio`
Thread priority.
- struct `context p_ctx`

- Processor context.*
- `thread_t * p_newer`
Newer registry element.
- `thread_t * p_older`
Older registry element.
- `const char * p_name`
Thread name or NULL.
- `stkalign_t * p_stklimit`
Thread stack boundary.
- `tstate_t p_state`
Current thread state.
- `tmode_t p_flags`
Various thread flags.
- `trefs_t p_refs`
References to this thread.
- `tslices_t p_preempt`
Number of ticks remaining to this thread.
- `volatile systime_t p_time`
Thread consumed time in ticks.
- union {
 - `msg_t rdymsg`
Thread wakeup code.
 - `msg_t exitcode`
Thread exit code.
 - `void * wtobjp`
Pointer to a generic "wait" object.
 - `thread_reference_t * wtrp`
Pointer to a generic thread reference object.
 - `struct ch_semaphore * wtsemp`
Pointer to a generic semaphore object.
 - `struct ch_mutex * wtmtxpx`
Pointer to a generic mutex object.
 - `eventmask_t ewmask`
Enabled events mask.
}
- State-specific fields.*
- `threads_list_t p_waiting`
Termination waiting list.
- `threads_queue_t p_msgqueue`
Messages queue.
- `msg_t p_msg`
Thread message.
- `eventmask_t p_epending`
Pending events mask.
- `struct ch_mutex * p_mtxlist`
List of the mutexes owned by this thread.
- `tprio_t p_realprio`
Thread's own, non-inherited, priority.
- `void * p_mpool`
Memory Pool where the thread workspace is returned.
- `time_measurement_t p_stats`
Thread statistics.

10.9.1 Detailed Description

Structure representing a thread.

Note

Not all the listed fields are always needed, by switching off some not needed ChibiOS/RT subsystems it is possible to save RAM space by shrinking this structure.

10.9.2 Field Documentation

10.9.2.1 `thread_t* ch_thread::p_next`

Next in the list/queue.

10.9.2.2 `thread_t* ch_thread::p_prev`

Previous in the queue.

10.9.2.3 `tprio_t ch_thread::p_prio`

Thread priority.

10.9.2.4 `struct context ch_thread::p_ctx`

Processor context.

10.9.2.5 `thread_t* ch_thread::p_newer`

Newer registry element.

10.9.2.6 `thread_t* ch_thread::p_older`

Older registry element.

10.9.2.7 `const char* ch_thread::p_name`

Thread name or NULL.

10.9.2.8 `stkalign_t* ch_thread::p_stklimit`

Thread stack boundary.

10.9.2.9 `tstate_t ch_thread::p_state`

Current thread state.

10.9.2.10 `tmode_t ch_thread::p_flags`

Various thread flags.

10.9.2.11 trefs_t ch_thread::p_refs

References to this thread.

10.9.2.12 tslices_t ch_thread::p_preempt

Number of ticks remaining to this thread.

10.9.2.13 volatile systime_t ch_thread::p_time

Thread consumed time in ticks.

Note

This field can overflow.

10.9.2.14 msg_t ch_thread::rdymsg

Thread wakeup code.

Note

This field contains the low level message sent to the thread by the waking thread or interrupt handler. The value is valid after exiting the [chSchWakeupS\(\)](#) function.

10.9.2.15 msg_t ch_thread::exitcode

Thread exit code.

Note

The thread termination code is stored in this field in order to be retrieved by the thread performing a [chThdWait\(\)](#) on this thread.

10.9.2.16 void* ch_thread::wtobjp

Pointer to a generic "wait" object.

Note

This field is used to get a generic pointer to a synchronization object and is valid when the thread is in one of the wait states.

10.9.2.17 thread_reference_t* ch_thread::wttrp

Pointer to a generic thread reference object.

Note

This field is used to get a pointer to a synchronization object and is valid when the thread is in [CH_STATE_SUSPENDED](#) state.

10.9.2.18 struct ch_semaphore* ch_thread::wtsemp

Pointer to a generic semaphore object.

Note

This field is used to get a pointer to a synchronization object and is valid when the thread is in CH_STATE↔_WTSEM state.

10.9.2.19 struct ch_mutex* ch_thread::wtmtx

Pointer to a generic mutex object.

Note

This field is used to get a pointer to a synchronization object and is valid when the thread is in CH_STATE↔_WTMTX state.

10.9.2.20 eventmask_t ch_thread::ewmask

Enabled events mask.

Note

This field is only valid while the thread is in the CH_STATE_WTOREVT or CH_STATE_WTANDEVT states.

10.9.2.21 union { ... } ch_thread::p_u

State-specific fields.

Note

All the fields declared in this union are only valid in the specified state or condition and are thus volatile.

10.9.2.22 threads_list_t ch_thread::p_waiting

Termination waiting list.

10.9.2.23 threads_queue_t ch_thread::p_msgqueue

Messages queue.

10.9.2.24 msg_t ch_thread::p_msg

Thread message.

10.9.2.25 eventmask_t ch_thread::p_epending

Pending events mask.

10.9.2.26 struct ch_mutex* ch_thread::p_mtxlist

List of the mutexes owned by this thread.

Note

The list is terminated by a NULL in this field.

10.9.2.27 tprio_t ch_thread::p_realprio

Thread's own, non-inherited, priority.

10.9.2.28 void* ch_thread::p_mpool

Memory Pool where the thread workspace is returned.

10.9.2.29 time_measurement_t ch_thread::p_stats

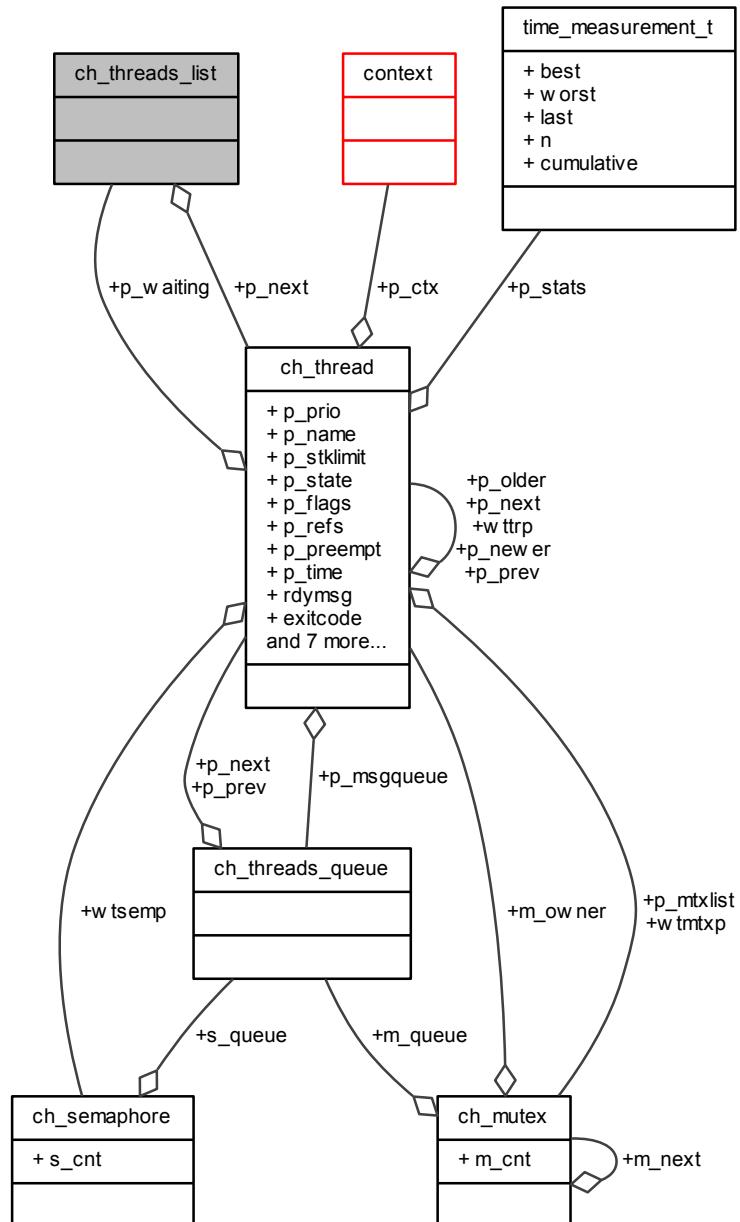
Thread statistics.

10.10 ch_threads_list Struct Reference

Generic threads single link list, it works like a stack.

```
#include <chsched.h>
```

Collaboration diagram for ch_threads_list:



Data Fields

- `thread_t * p_next`

Next in the list/queue.

10.10.1 Detailed Description

Generic threads single link list, it works like a stack.

10.10.2 Field Documentation

10.10.2.1 `thread_t* ch_threads_list::p_next`

Next in the list/queue.

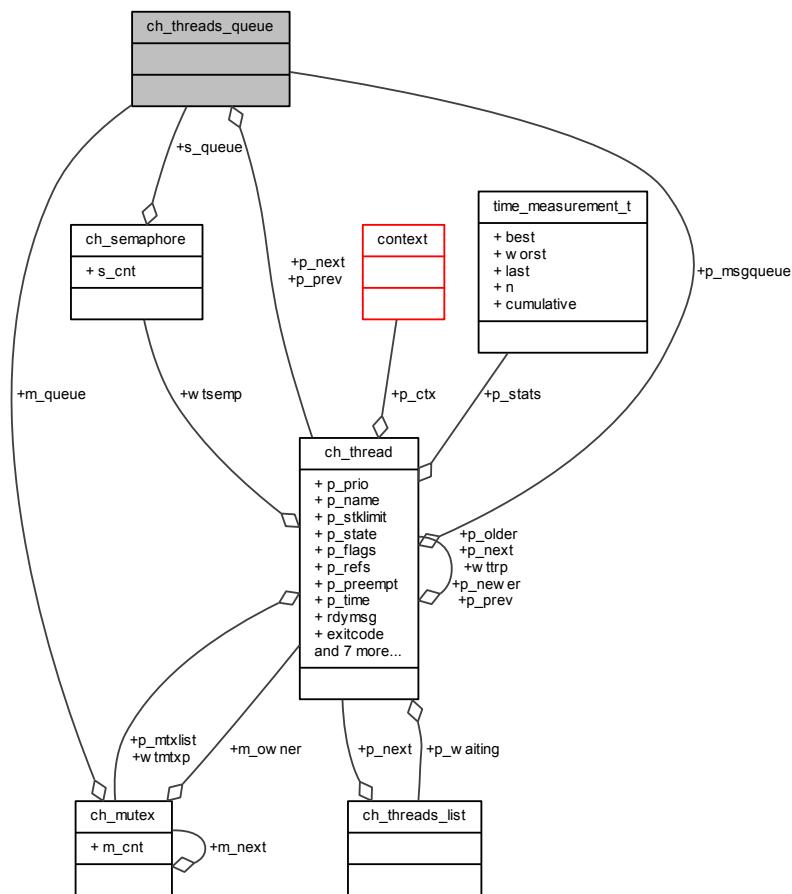
10.11 ch_threads_queue Struct Reference

Generic threads bidirectional linked list header and element.

```
#include <chsched.h>
```

Inherited by `ch_ready_list`.

Collaboration diagram for `ch_threads_queue`:



Data Fields

- `thread_t * p_next`

Next in the list/queue.

- `thread_t * p_prev`

Previous in the queue.

10.11.1 Detailed Description

Generic threads bidirectional linked list header and element.

10.11.2 Field Documentation

10.11.2.1 `thread_t* ch_threads_queue::p_next`

Next in the list/queue.

10.11.2.2 `thread_t* ch_threads_queue::p_prev`

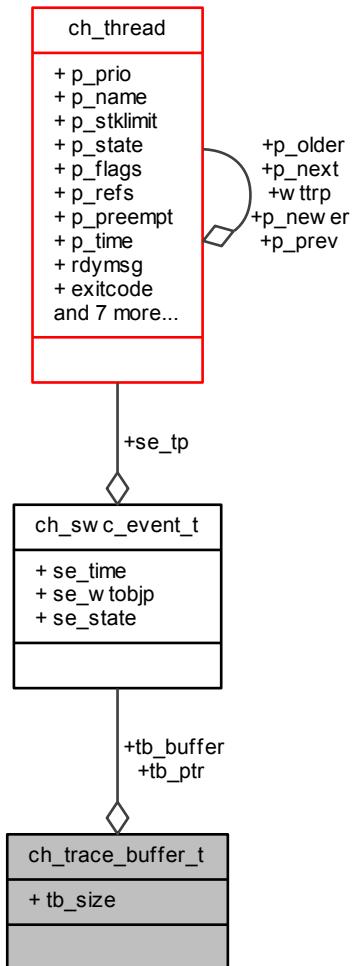
Previous in the queue.

10.12 ch_trace_buffer_t Struct Reference

Trace buffer header.

```
#include <chdebug.h>
```

Collaboration diagram for ch_trace_buffer_t:



Data Fields

- unsigned `tb_size`
Trace buffer size (entries).
- `ch_sw_c_event_t * tb_ptr`
Pointer to the buffer front.
- `ch_sw_c_event_t tb_buffer [CH_DBG_TRACE_BUFFER_SIZE]`
Ring buffer.

10.12.1 Detailed Description

Trace buffer header.

10.12.2 Field Documentation

10.12.2.1 unsigned ch_trace_buffer_t::tb_size

Trace buffer size (entries).

10.12.2.2 ch_swc_event_t* ch_trace_buffer_t::tb_ptr

Pointer to the buffer front.

10.12.2.3 ch_swc_event_t ch_trace_buffer_t::tb_buffer[CH_DBG_TRACE_BUFFER_SIZE]

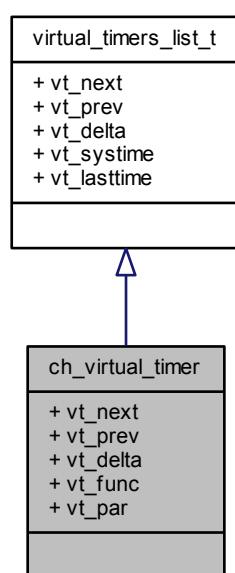
Ring buffer.

10.13 ch_virtual_timer Struct Reference

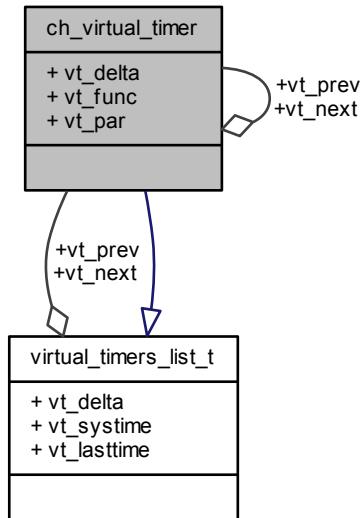
Virtual Timer descriptor structure.

```
#include <chsched.h>
```

Inheritance diagram for ch_virtual_timer:



Collaboration diagram for ch_virtual_timer:



Data Fields

- `virtual_timer_t * vt_next`
Next timer in the list.
- `virtual_timer_t * vt_prev`
Previous timer in the list.
- `systime_t vt_delta`
Time delta before timeout.
- `vfunc_t vt_func`
Timer callback function pointer.
- `void * vt_par`
Timer callback function parameter.

10.13.1 Detailed Description

Virtual Timer descriptor structure.

10.13.2 Field Documentation

10.13.2.1 `virtual_timer_t* ch_virtual_timer::vt_next`

Next timer in the list.

10.13.2.2 `virtual_timer_t* ch_virtual_timer::vt_prev`

Previous timer in the list.

10.13.2.3 systime_t ch_virtual_timer::vt_delta

Time delta before timeout.

10.13.2.4 vfunc_t ch_virtual_timer::vt_func

Timer callback function pointer.

10.13.2.5 void* ch_virtual_timer::vt_par

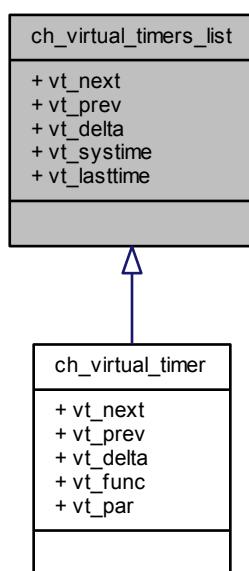
Timer callback function parameter.

10.14 ch_virtual_timers_list Struct Reference

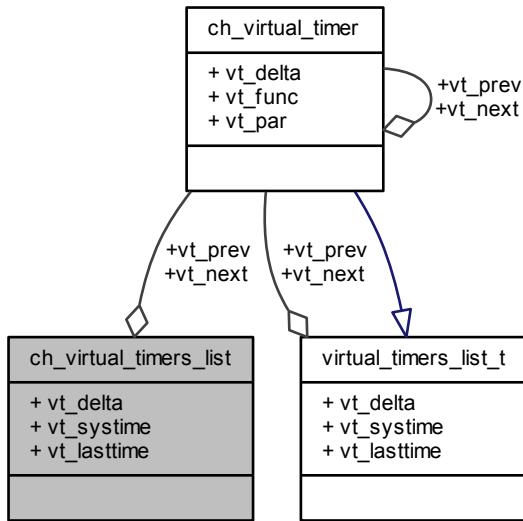
Virtual timers list header.

```
#include <chsched.h>
```

Inheritance diagram for ch_virtual_timers_list:



Collaboration diagram for ch_virtual_timers_list:



Data Fields

- `virtual_timer_t * vt_next`
Next timer in the delta list.
- `virtual_timer_t * vt_prev`
Last timer in the delta list.
- `systime_t vt_delta`
Must be initialized to -1.
- `volatile systime_t vt_systime`
System Time counter.
- `systime_t vt_lasttime`
System time of the last tick event.

10.14.1 Detailed Description

Virtual timers list header.

Note

The timers list is implemented as a double link bidirectional list in order to make the unlink time constant, the reset of a virtual timer is often used in the code.

10.14.2 Field Documentation

10.14.2.1 `virtual_timer_t* ch_virtual_timers_list::vt_next`

Next timer in the delta list.

10.14.2.2 virtual_timer_t* ch_virtual_timers_list::vt_prev

Last timer in the delta list.

10.14.2.3 systime_t ch_virtual_timers_list::vt_delta

Must be initialized to -1.

10.14.2.4 volatile systime_t ch_virtual_timers_list::vt_systime

System Time counter.

10.14.2.5 systime_t ch_virtual_timers_list::vt_lasttime

System time of the last tick event.

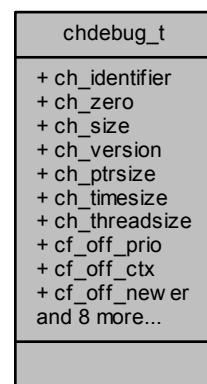
System time of the last tick event.

10.15 chdebug_t Struct Reference

ChibiOS/RT memory signature record.

```
#include <chregistry.h>
```

Collaboration diagram for chdebug_t:



Data Fields

- char **ch_identifier** [4]

Always set to "main".

- uint8_t **ch_zero**

Must be zero.

- uint8_t **ch_size**

Size of this structure.

- `uint16_t ch_version`
Encoded ChibiOS/RT version.
- `uint8_t ch_ptrsize`
Size of a pointer.
- `uint8_t ch_timesize`
Size of a systime_t.
- `uint8_t ch_threadsize`
Size of a thread_t.
- `uint8_t cf_off_prio`
Offset of p_prio field.
- `uint8_t cf_off_ctx`
Offset of p_ctx field.
- `uint8_t cf_off_newer`
Offset of p_newer field.
- `uint8_t cf_off_older`
Offset of p_older field.
- `uint8_t cf_off_name`
Offset of p_name field.
- `uint8_t cf_off_stklimit`
Offset of p_stklimit field.
- `uint8_t cf_off_state`
Offset of p_state field.
- `uint8_t cf_off_flags`
Offset of p_flags field.
- `uint8_t cf_off_refs`
Offset of p_refs field.
- `uint8_t cf_off_preempt`
Offset of p_preempt field.
- `uint8_t cf_off_time`
Offset of p_time field.

10.15.1 Detailed Description

ChibiOS/RT memory signature record.

10.15.2 Field Documentation

10.15.2.1 `char chdebug_t::ch_identifier[4]`

Always set to "main".

10.15.2.2 `uint8_t chdebug_t::ch_zero`

Must be zero.

10.15.2.3 `uint8_t chdebug_t::ch_size`

Size of this structure.

10.15.2.4 `uint16_t chdebug_t::ch_version`

Encoded ChibiOS/RT version.

10.15.2.5 `uint8_t chdebug_t::ch_ptrsize`

Size of a pointer.

10.15.2.6 `uint8_t chdebug_t::ch_timesize`

Size of a `systime_t`.

10.15.2.7 `uint8_t chdebug_t::ch_threadsize`

Size of a `thread_t`.

10.15.2.8 `uint8_t chdebug_t::cf_off_prio`

Offset of `p_prio` field.

10.15.2.9 `uint8_t chdebug_t::cf_off_ctx`

Offset of `p_ctx` field.

10.15.2.10 `uint8_t chdebug_t::cf_off_newer`

Offset of `p_newer` field.

10.15.2.11 `uint8_t chdebug_t::cf_off_older`

Offset of `p_older` field.

10.15.2.12 `uint8_t chdebug_t::cf_off_name`

Offset of `p_name` field.

10.15.2.13 `uint8_t chdebug_t::cf_off_stklimit`

Offset of `p_stklimit` field.

10.15.2.14 `uint8_t chdebug_t::cf_off_state`

Offset of `p_state` field.

10.15.2.15 `uint8_t chdebug_t::cf_off_flags`

Offset of `p_flags` field.

10.15.2.16 uint8_t chdebug_t::cf_off_refs

Offset of p_refs field.

10.15.2.17 uint8_t chdebug_t::cf_off_preempt

Offset of p_preempt field.

10.15.2.18 uint8_t chdebug_t::cf_off_time

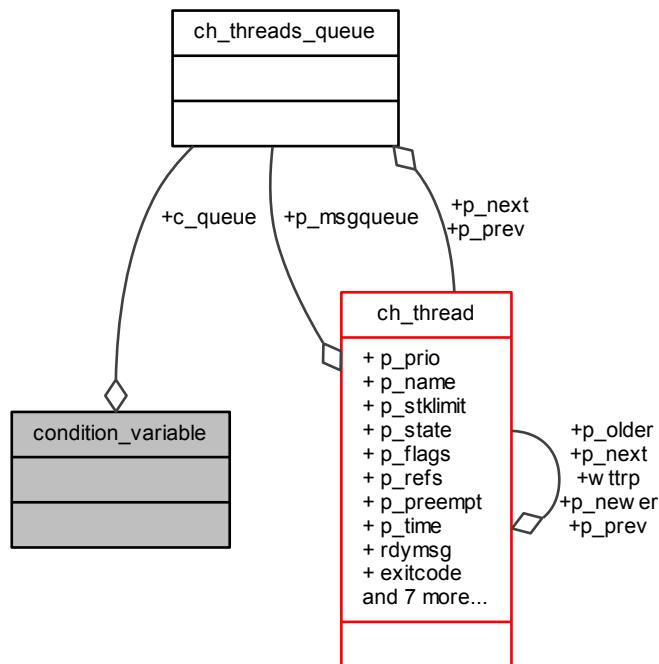
Offset of p_time field.

10.16 condition_variable Struct Reference

condition_variable_t structure.

```
#include <chcond.h>
```

Collaboration diagram for condition_variable:



Data Fields

- `threads_queue_t c_queue`

Condition variable threads queue.

10.16.1 Detailed Description

condition_variable_t structure.

10.16.2 Field Documentation

10.16.2.1 threads_queue_t condition_variable::c_queue

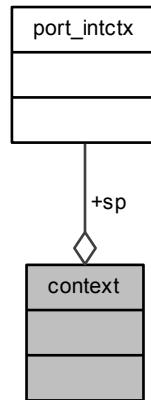
Condition variable threads queue.

10.17 context Struct Reference

Platform dependent part of the thread_t structure.

```
#include <chcore.h>
```

Collaboration diagram for context:



10.17.1 Detailed Description

Platform dependent part of the thread_t structure.

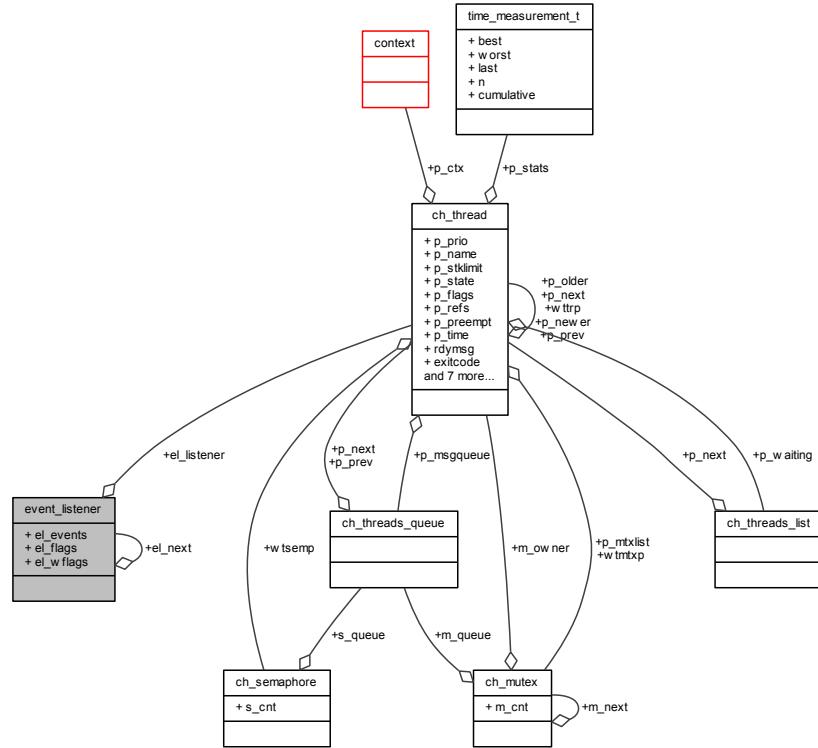
This structure usually contains just the saved stack pointer defined as a pointer to a `port_intctx` structure.

10.18 event_listener Struct Reference

Event Listener structure.

```
#include <chevents.h>
```

Collaboration diagram for event_listener:



Data Fields

- `event_listener_t * el_next`
Next Event Listener registered on the event source.
- `thread_t * el_listener`
Thread interested in the event source.
- `eventmask_t el_events`
Events to be set in the listening thread.
- `eventflags_t el_flags`
Flags added to the listener by the event source.
- `eventflags_t el_wflags`
Flags that this listener interested in.

10.18.1 Detailed Description

Event Listener structure.

10.18.2 Field Documentation

10.18.2.1 `event_listener_t * event_listener::el_next`

Next Event Listener registered on the event source.

10.18.2.2 thread_t* event_listener::el_listener

Thread interested in the event source.

10.18.2.3 eventmask_t event_listener::el_events

Events to be set in the listening thread.

10.18.2.4 eventflags_t event_listener::el_flags

Flags added to the listener by the event source.

10.18.2.5 eventflags_t event_listener::el_wflags

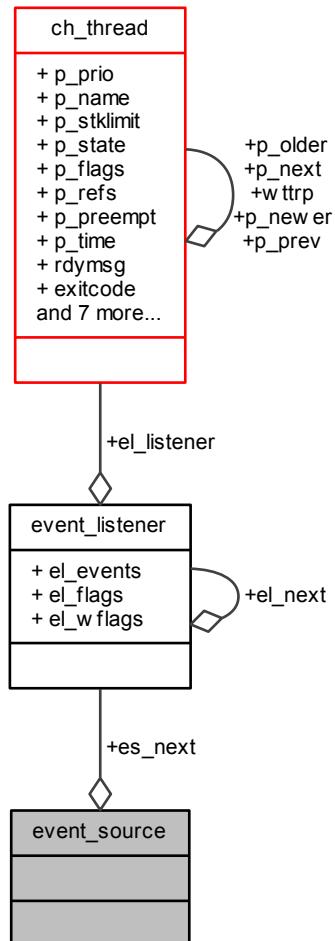
Flags that this listener interested in.

10.19 event_source Struct Reference

Event Source structure.

```
#include <chevents.h>
```

Collaboration diagram for event_source:



Data Fields

- `event_listener_t * es_next`

First Event Listener registered on the Event Source.

10.19.1 Detailed Description

Event Source structure.

10.19.2 Field Documentation

10.19.2.1 `event_listener_t* event_source::es_next`

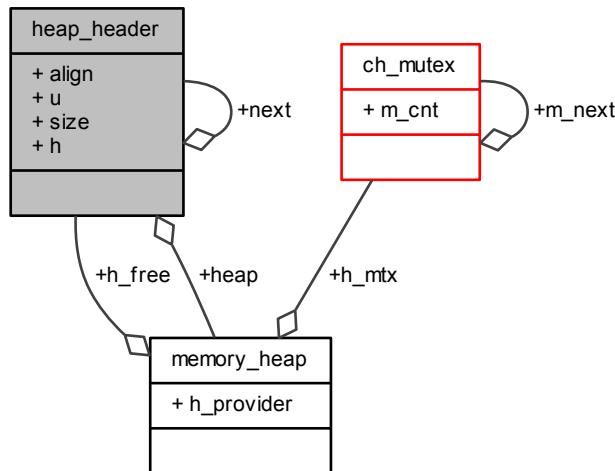
First Event Listener registered on the Event Source.

10.20 heap_header Union Reference

Memory heap block header.

```
#include <chheap.h>
```

Collaboration diagram for heap_header:



10.20.1 Detailed Description

Memory heap block header.

10.20.2 Field Documentation

10.20.2.1 union heap_header* heap_header::next

Next block in free list.

10.20.2.2 memory_heap_t* heap_header::heap

Block owner heap.

10.20.2.3 union { ... } heap_header::u

Overlapped fields.

10.20.2.4 size_t heap_header::size

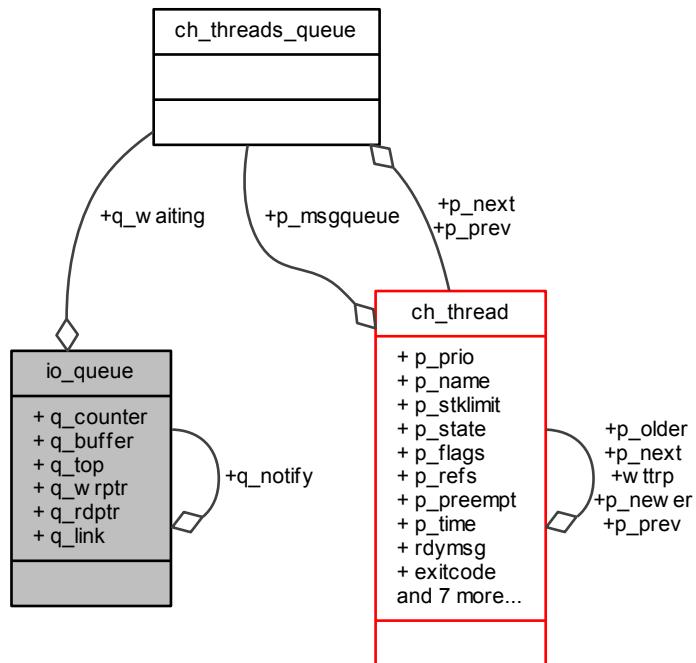
Size of the memory block.

10.21 io_queue Struct Reference

Generic I/O queue structure.

```
#include <chqueues.h>
```

Collaboration diagram for io_queue:



Data Fields

- **threads_queue_t q_waiting**
Queue of waiting threads.
- volatile size_t **q_counter**
Resources counter.
- uint8_t * **q_buffer**
Pointer to the queue buffer.
- uint8_t * **q_top**
Pointer to the first location after the buffer.
- uint8_t * **q_wptr**
Write pointer.
- uint8_t * **q_rptr**
Read pointer.
- qnotify_t **q_notify**
Data notification callback.
- void * **q_link**
Application defined field.

10.21.1 Detailed Description

Generic I/O queue structure.

This structure represents a generic Input or Output asymmetrical queue. The queue is asymmetrical because one end is meant to be accessed from a thread context, and thus can be blocking, the other end is accessible from interrupt handlers or from within a kernel lock zone (see **I-Locked** and **S-Locked** states in [System States](#)) and is non-blocking.

10.21.2 Field Documentation

10.21.2.1 threads_queue_t io_queue::q_waiting

Queue of waiting threads.

10.21.2.2 volatile size_t io_queue::q_counter

Resources counter.

10.21.2.3 uint8_t* io_queue::q_buffer

Pointer to the queue buffer.

10.21.2.4 uint8_t* io_queue::q_top

Pointer to the first location after the buffer.

10.21.2.5 uint8_t* io_queue::q_wptr

Write pointer.

10.21.2.6 uint8_t* io_queue::q_rdptr

Read pointer.

10.21.2.7 qnotify_t io_queue::q_notify

Data notification callback.

10.21.2.8 void* io_queue::q_link

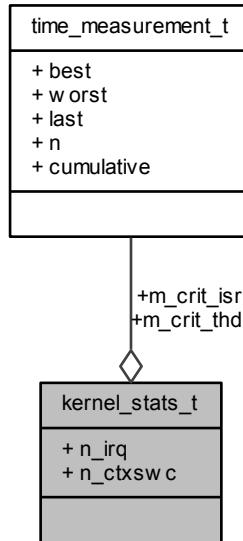
Application defined field.

10.22 kernel_stats_t Struct Reference

Type of a kernel statistics structure.

```
#include <chstats.h>
```

Collaboration diagram for kernel_stats_t:



Data Fields

- [ucnt_t n_irq](#)
Number of IRQs.
- [ucnt_t n_ctxswc](#)
Number of context switches.
- [time_measurement_t m_crit_thd](#)
Measurement of threads critical zones duration.
- [time_measurement_t m_crit_isr](#)
Measurement of ISRs critical zones duration.

10.22.1 Detailed Description

Type of a kernel statistics structure.

10.22.2 Field Documentation

10.22.2.1 ucnt_t kernel_stats_t::n_irq

Number of IRQs.

10.22.2.2 ucnt_t kernel_stats_t::n_ctxswc

Number of context switches.

10.22.2.3 time_measurement_t kernel_stats_t::m_crit_thd

Measurement of threads critical zones duration.

10.22.2.4 time_measurement_t kernel_stats_t::m_crit_isr

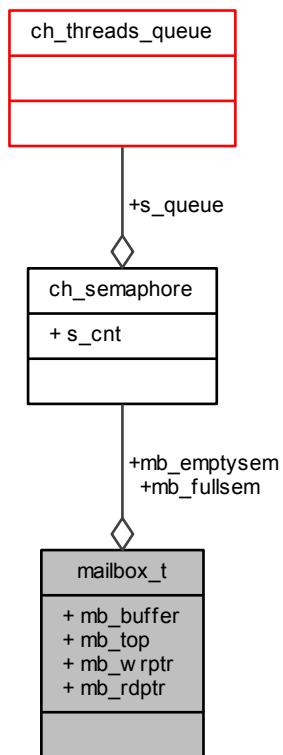
Measurement of ISRs critical zones duration.

10.23 mailbox_t Struct Reference

Structure representing a mailbox object.

```
#include <chmboxes.h>
```

Collaboration diagram for mailbox_t:



Data Fields

- `msg_t * mb_buffer`
Pointer to the mailbox buffer.
- `msg_t * mb_top`
Pointer to the location after the buffer.
- `msg_t * mb_wrptr`

Write pointer.

- `msg_t * mb_rptr`

Read pointer.

- `semaphore_t mb_fullsem`

Full counter semaphore_t.

- `semaphore_t mb_emptysem`

Empty counter semaphore_t.

10.23.1 Detailed Description

Structure representing a mailbox object.

10.23.2 Field Documentation

10.23.2.1 `msg_t* mailbox_t::mb_buffer`

Pointer to the mailbox buffer.

10.23.2.2 `msg_t* mailbox_t::mb_top`

Pointer to the location after the buffer.

10.23.2.3 `msg_t* mailbox_t::mb_wptr`

Write pointer.

10.23.2.4 `msg_t* mailbox_t::mb_rptr`

Read pointer.

10.23.2.5 `semaphore_t mailbox_t::mb_fullsem`

Full counter semaphore_t.

10.23.2.6 `semaphore_t mailbox_t::mb_emptysem`

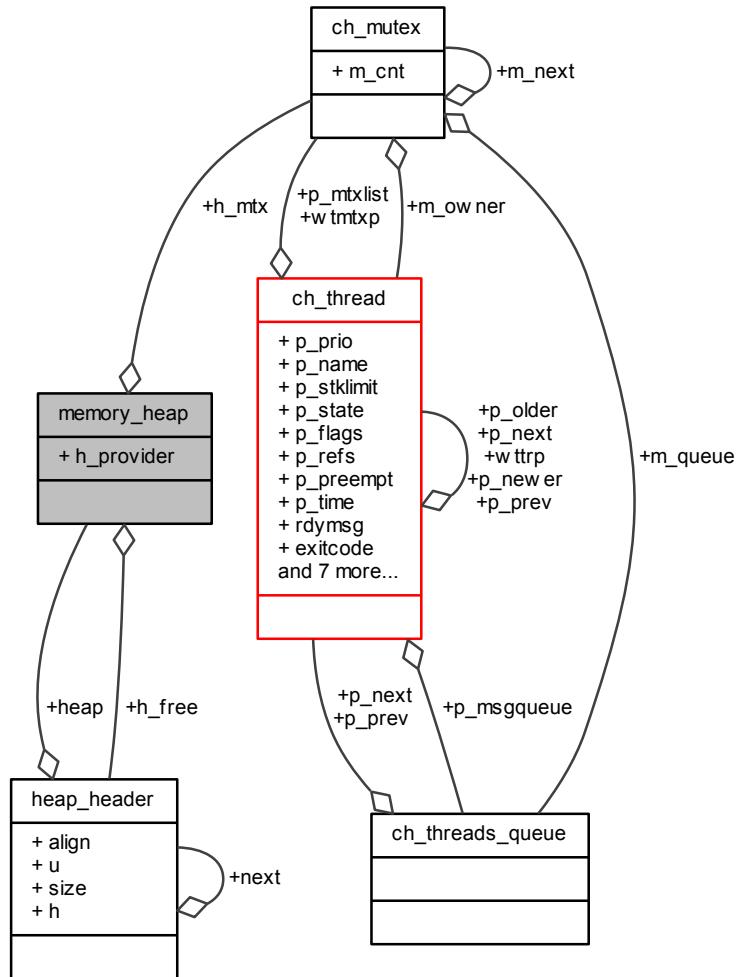
Empty counter semaphore_t.

10.24 memory_heap Struct Reference

Structure describing a memory heap.

```
#include <chheap.h>
```

Collaboration diagram for memory_heap:



Data Fields

- `memgetfunc_t h_provider`
Memory blocks provider for this heap.
- union `heap_header h_free`
Free blocks list header.
- `mutex_t h_mtx`
Heap access mutex.

10.24.1 Detailed Description

Structure describing a memory heap.

10.24.2 Field Documentation

10.24.2.1 memgetfunc_t memory_heap::h_provider

Memory blocks provider for this heap.

10.24.2.2 union heap_header memory_heap::h_free

Free blocks list header.

10.24.2.3 mutex_t memory_heap::h_mtx

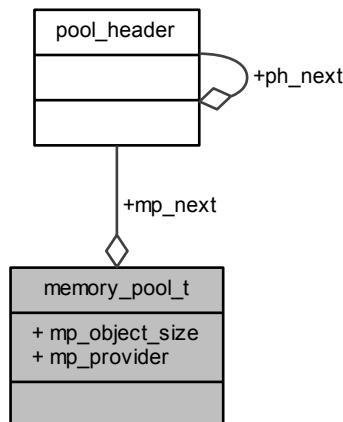
Heap access mutex.

10.25 memory_pool_t Struct Reference

Memory pool descriptor.

```
#include <chmempools.h>
```

Collaboration diagram for memory_pool_t:



Data Fields

- struct [pool_header * mp_next](#)

Pointer to the header.

- size_t [mp_object_size](#)

Memory pool objects size.

- [memgetfunc_t mp_provider](#)

Memory blocks provider for this pool.

10.25.1 Detailed Description

Memory pool descriptor.

10.25.2 Field Documentation

10.25.2.1 struct pool_header* memory_pool_t::mp_next

Pointer to the header.

10.25.2.2 size_t memory_pool_t::mp_object_size

Memory pool objects size.

10.25.2.3 memgetfunc_t memory_pool_t::mp_provider

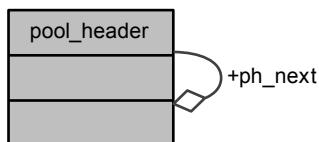
Memory blocks provider for this pool.

10.26 pool_header Struct Reference

Memory pool free object header.

```
#include <chmempools.h>
```

Collaboration diagram for pool_header:



Data Fields

- struct [pool_header * ph_next](#)

Pointer to the next pool header in the list.

10.26.1 Detailed Description

Memory pool free object header.

10.26.2 Field Documentation

10.26.2.1 struct pool_header* pool_header::ph_next

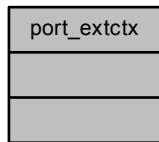
Pointer to the next pool header in the list.

10.27 port_extctx Struct Reference

Interrupt saved context.

```
#include <chcore.h>
```

Collaboration diagram for port_extctx:



10.27.1 Detailed Description

Interrupt saved context.

This structure represents the stack frame saved during a preemption-capable interrupt handler.

Note

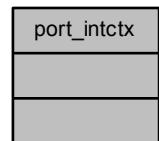
R2 and R13 are not saved because those are assumed to be immutable during the system life cycle.

10.28 port_intctx Struct Reference

System saved context.

```
#include <chcore.h>
```

Collaboration diagram for port_intctx:



10.28.1 Detailed Description

System saved context.

This structure represents the inner stack frame during a context switching.

Note

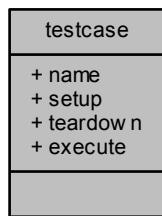
R2 and R13 are not saved because those are assumed to be immutable during the system life cycle.
LR is stored in the caller context so it is not present in this structure.

10.29 testcase Struct Reference

Structure representing a test case.

```
#include <test.h>
```

Collaboration diagram for testcase:



Data Fields

- `const char * name`
Test case name.
- `void(* setup)(void)`
Test case preparation function.
- `void(* teardown)(void)`
Test case clean up function.
- `void(* execute)(void)`
Test case execution function.

10.29.1 Detailed Description

Structure representing a test case.

10.29.2 Field Documentation

10.29.2.1 `const char* testcase::name`

Test case name.

10.29.2.2 `void(* testcase::setup)(void)`

Test case preparation function.

10.29.2.3 void(* testcase::teardown) (void)

Test case clean up function.

10.29.2.4 void(* testcase::execute) (void)

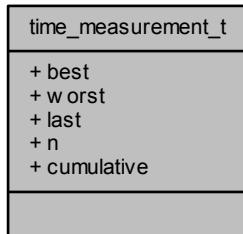
Test case execution function.

10.30 time_measurement_t Struct Reference

Type of a Time Measurement object.

```
#include <chtm.h>
```

Collaboration diagram for time_measurement_t:



Data Fields

- [rtcnt_t best](#)
Best measurement.
- [rtcnt_t worst](#)
Worst measurement.
- [rtcnt_t last](#)
Last measurement.
- [ucnt_t n](#)
Number of measurements.
- [rttime_t cumulative](#)
Cumulative measurement.

10.30.1 Detailed Description

Type of a Time Measurement object.

Note

The maximum measurable time period depends on the implementation of the realtime counter and its clock frequency.

The measurement is not 100% cycle-accurate, it can be in excess of few cycles depending on the compiler and target architecture.

Interrupts can affect measurement if the measurement is performed with interrupts enabled.

10.30.2 Field Documentation

10.30.2.1 `rtcnt_t time_measurement_t::best`

Best measurement.

10.30.2.2 `rtcnt_t time_measurement_t::worst`

Worst measurement.

10.30.2.3 `rtcnt_t time_measurement_t::last`

Last measurement.

10.30.2.4 `ucnt_t time_measurement_t::n`

Number of measurements.

10.30.2.5 `rttime_t time_measurement_t::cumulative`

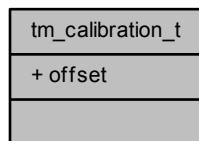
Cumulative measurement.

10.31 tm_calibration_t Struct Reference

Type of a time measurement calibration data.

```
#include <chtm.h>
```

Collaboration diagram for tm_calibration_t:



Data Fields

- `rtcnt_t offset`

Measurement calibration value.

10.31.1 Detailed Description

Type of a time measurement calibration data.

10.31.2 Field Documentation

10.31.2.1 `rtcnt_t tm_calibration_t::offset`

Measurement calibration value.

Chapter 11

File Documentation

11.1 ch.h File Reference

ChibiOS/RT main include file.

```
#include "ctypes.h"
#include "chconf.h"
#include "chlicense.h"
#include "chsystypes.h"
#include "chcore.h"
#include "chdebug.h"
#include "chtm.h"
#include "chstats.h"
#include "chsched.h"
#include "chsys.h"
#include "chvt.h"
#include "chthreads.h"
#include "chregistry.h"
#include "chsem.h"
#include "chbsem.h"
#include "chmtx.h"
#include "chcond.h"
#include "chevents.h"
#include "chmsg.h"
#include "chmboxes.h"
#include "chmemcore.h"
#include "chheap.h"
#include "chmempools.h"
#include "chdynamic.h"
#include "chqueues.h"
#include "chstreams.h"
```

Macros

- #define **_CHIBIOS_RT_**
ChibiOS/RT identification macro.
- #define **CH_KERNEL_STABLE** 1
Stable release flag.

ChibiOS/RT version identification

- `#define CH_KERNEL_VERSION "3.1.5"`
Kernel version string.
- `#define CH_KERNEL_MAJOR 3`
Kernel version major number.
- `#define CH_KERNEL_MINOR 1`
Kernel version minor number.
- `#define CH_KERNEL_PATCH 5`
Kernel version patch number.

11.1.1 Detailed Description

ChibiOS/RT main include file.

This header includes all the required kernel headers so it is the only kernel header you usually want to include in your application.

11.2 chbsem.h File Reference

Binary semaphores structures and macros.

Data Structures

- struct `binary_semaphore_t`
Binary semaphore type.

Macros

- `#define _BSEMAPHORE_DATA(name, taken) { _SEMAPHORE_DATA(name.bs_sem, ((taken) ? 0 : 1))}`
Data part of a static semaphore initializer.
- `#define BSEMAPHORE_DECL(name, taken) binary_semaphore_t name = _BSEMAPHORE_DATA(name, taken)`
Static semaphore initializer.

Functions

- static void `chBSemObjectInit (binary_semaphore_t *bsp, bool taken)`
Initializes a binary semaphore.
- static `msg_t chBSemWait (binary_semaphore_t *bsp)`
Wait operation on the binary semaphore.
- static `msg_t chBSemWaitS (binary_semaphore_t *bsp)`
Wait operation on the binary semaphore.
- static `msg_t chBSemWaitTimeoutS (binary_semaphore_t *bsp, systime_t time)`
Wait operation on the binary semaphore.
- static `msg_t chBSemWaitTimeout (binary_semaphore_t *bsp, systime_t time)`
Wait operation on the binary semaphore.
- static void `chBSemResetl (binary_semaphore_t *bsp, bool taken)`
Reset operation on the binary semaphore.
- static void `chBSemReset (binary_semaphore_t *bsp, bool taken)`
Reset operation on the binary semaphore.
- static void `chBSemSignall (binary_semaphore_t *bsp)`
Performs a signal operation on a binary semaphore.

- static void `chBSemSignal (binary_semaphore_t *bsp)`
Performs a signal operation on a binary semaphore.
- static bool `chBSemGetState (binary_semaphore_t *bsp)`
Returns the binary semaphore current state.

11.2.1 Detailed Description

Binary semaphores structures and macros.

11.3 chcond.c File Reference

Condition Variables code.

```
#include "ch.h"
```

Functions

- void `chCondObjectInit (condition_variable_t *cp)`
Initializes a condition_variable_t structure.
- void `chCondSignal (condition_variable_t *cp)`
Signals one thread that is waiting on the condition variable.
- void `chCondSignall (condition_variable_t *cp)`
Signals one thread that is waiting on the condition variable.
- void `chCondBroadcast (condition_variable_t *cp)`
Signals all threads that are waiting on the condition variable.
- void `chCondBroadcastl (condition_variable_t *cp)`
Signals all threads that are waiting on the condition variable.
- msg_t `chCondWait (condition_variable_t *cp)`
Waits on the condition variable releasing the mutex lock.
- msg_t `chCondWaitS (condition_variable_t *cp)`
Waits on the condition variable releasing the mutex lock.
- msg_t `chCondWaitTimeout (condition_variable_t *cp, systime_t time)`
Waits on the condition variable releasing the mutex lock.
- msg_t `chCondWaitTimeoutS (condition_variable_t *cp, systime_t time)`
Waits on the condition variable releasing the mutex lock.

11.3.1 Detailed Description

Condition Variables code.

11.4 chcond.h File Reference

Condition Variables macros and structures.

Data Structures

- struct `condition_variable`
condition_variable_t structure.

Macros

- `#define _CONDVAR_DATA(name) {_THREADS_QUEUE_DATA(name.c_queue)}`
Data part of a static condition variable initializer.
- `#define CONDVAR_DECL(name) condition_variable_t name = _CONDVAR_DATA(name)`
Static condition variable initializer.

Typedefs

- `typedef struct condition_variable condition_variable_t`
condition_variable_t structure.

Functions

- `void chCondObjectInit (condition_variable_t *cp)`
Initializes a condition_variable_t structure.
- `void chCondSignal (condition_variable_t *cp)`
Signals one thread that is waiting on the condition variable.
- `void chCondSignall (condition_variable_t *cp)`
Signals one thread that is waiting on the condition variable.
- `void chCondBroadcast (condition_variable_t *cp)`
Signals all threads that are waiting on the condition variable.
- `void chCondBroadcastl (condition_variable_t *cp)`
Signals all threads that are waiting on the condition variable.
- `msg_t chCondWait (condition_variable_t *cp)`
Waits on the condition variable releasing the mutex lock.
- `msg_t chCondWaitS (condition_variable_t *cp)`
Waits on the condition variable releasing the mutex lock.
- `msg_t chCondWaitTimeout (condition_variable_t *cp, systime_t time)`
Waits on the condition variable releasing the mutex lock.
- `msg_t chCondWaitTimeoutS (condition_variable_t *cp, systime_t time)`
Waits on the condition variable releasing the mutex lock.

11.4.1 Detailed Description

Condition Variables macros and structures.

11.5 chconf.h File Reference

Configuration file template.

Macros

System timers settings

- `#define CH_CFG_ST_RESOLUTION 32`
System time counter resolution.
- `#define CH_CFG_ST_FREQUENCY 10000`
System tick frequency.
- `#define CH_CFG_ST_TIMEDELTA 2`

Time delta constant for the tick-less mode.

Kernel parameters and options

- #define CH_CFG_TIME_QUANTUM 0
Round robin interval.
- #define CH_CFG_MEMCORE_SIZE 0
Managed RAM size.
- #define CH_CFG_NO_IDLE_THREAD FALSE
Idle thread automatic spawn suppression.

Performance options

- #define CH_CFG_OPTIMIZE_SPEED TRUE
OS optimization.

Subsystem options

- #define CH_CFG_USE_TM TRUE
Time Measurement APIs.
- #define CH_CFG_USE_REGISTRY TRUE
Threads registry APIs.
- #define CH_CFG_USE_WAITEXIT TRUE
Threads synchronization APIs.
- #define CH_CFG_USE_SEMAPHORES TRUE
Semaphores APIs.
- #define CH_CFG_USE_SEMAPHORES_PRIORITY FALSE
Semaphores queuing mode.
- #define CH_CFG_USE_MUTEXES TRUE
Mutexes APIs.
- #define CH_CFG_USE_MUTEXES_RECURSIVE FALSE
Enables recursive behavior on mutexes.
- #define CH_CFG_USE_CONDVARS TRUE
Conditional Variables APIs.
- #define CH_CFG_USE_CONDVARS_TIMEOUT TRUE
Conditional Variables APIs with timeout.
- #define CH_CFG_USE_EVENTS TRUE
Events Flags APIs.
- #define CH_CFG_USE_EVENTS_TIMEOUT TRUE
Events Flags APIs with timeout.
- #define CH_CFG_USE_MESSAGES TRUE
Synchronous Messages APIs.
- #define CH_CFG_USE_MESSAGES_PRIORITY FALSE
Synchronous Messages queuing mode.
- #define CH_CFG_USE_MAILBOXES TRUE
Mailboxes APIs.
- #define CH_CFG_USE_QUEUES TRUE
I/O Queues APIs.
- #define CH_CFG_USE_MEMCORE TRUE
Core Memory Manager APIs.
- #define CH_CFG_USE_HEAP TRUE
Heap Allocator APIs.
- #define CH_CFG_USE_MEMPOOLS TRUE
Memory Pools Allocator APIs.
- #define CH_CFG_USE_DYNAMIC TRUE
Dynamic Threads APIs.

Debug options

- `#define CH_DBG_STATISTICS FALSE`
Debug option, kernel statistics.
- `#define CH_DBG_SYSTEM_STATE_CHECK FALSE`
Debug option, system state check.
- `#define CH_DBG_ENABLE_CHECKS FALSE`
Debug option, parameters checks.
- `#define CH_DBG_ENABLE_ASSERTS FALSE`
Debug option, consistency checks.
- `#define CH_DBG_ENABLE_TRACE FALSE`
Debug option, trace buffer.
- `#define CH_DBG_ENABLE_STACK_CHECK FALSE`
Debug option, stack checks.
- `#define CH_DBG_FILL_THREADS FALSE`
Debug option, stacks initialization.
- `#define CH_DBG_THREADS_PROFILING FALSE`
Debug option, threads profiling.

Kernel hooks

- `#define CH_CFG_THREAD_EXTRA_FIELDS /* Add threads custom fields here. */`
Threads descriptor structure extension.
- `#define CH_CFG_THREAD_INIT_HOOK(tp)`
Threads initialization hook.
- `#define CH_CFG_THREAD_EXIT_HOOK(tp)`
Threads finalization hook.
- `#define CH_CFG_CONTEXT_SWITCH_HOOK(ntp, otp)`
Context switch hook.
- `#define CH_CFG_IDLE_ENTER_HOOK()`
Idle thread enter hook.
- `#define CH_CFG_IDLE_LEAVE_HOOK()`
Idle thread leave hook.
- `#define CH_CFG_IDLE_LOOP_HOOK()`
Idle Loop hook.
- `#define CH_CFG_SYSTEM_TICK_HOOK()`
System tick event hook.
- `#define CH_CFG_SYSTEM_HALT_HOOK(reason)`
System halt hook.

11.5.1 Detailed Description

Configuration file template.

A copy of this file must be placed in each project directory, it contains the application specific kernel settings.

11.6 chcore.c File Reference

Port related template code.

```
#include "ch.h"
```

Functions

- `void __port_init (void)`
Port-related initialization code.
- `void __port_switch (thread_t *ntp, thread_t *otp)`
Performs a context switch between two threads.

11.6.1 Detailed Description

Port related template code.

This file is a template of the system driver functions provided by a port. Some of the following functions may be implemented as macros in [chcore.h](#) if the implementer decides that there is an advantage in doing so, for example because performance concerns.

11.7 chcore.h File Reference

Port related template macros and structures.

Data Structures

- struct [port_extctx](#)
Interrupt saved context.
- struct [port_intctx](#)
System saved context.
- struct [context](#)
Platform dependent part of the `thread_t` structure.

Macros

- #define [PORT_IDLE_THREAD_STACK_SIZE](#) 32
Stack size for the system idle thread.
- #define [PORT_INT_REQUIRED_STACK](#) 256
Per-thread stack overhead for interrupts servicing.
- #define [PORT_USE_ALT_TIMER](#) FALSE
Enables an alternative timer implementation.
- #define [PORT_SETUP_CONTEXT](#)(tp, workspace, wsize, pf, arg)
Platform dependent part of the `chThdCreateI()` API.
- #define [PORT_WA_SIZE](#)(n)
Computes the thread working area global size.
- #define [PORT_IRQ_IS_VALID_PRIORITY](#)(n) false
Priority level verification macro.
- #define [PORT_IRQ_IS_VALID_KERNEL_PRIORITY](#)(n) false
Priority level verification macro.
- #define [PORT_IRQ_PROLOGUE](#)()
IRQ prologue code.
- #define [PORT_IRQ_EPILOGUE](#)()
IRQ epilogue code.
- #define [PORT_IRQ_HANDLER](#)(id) void id(void)
IRQ handler function declaration.
- #define [PORT_FAST_IRQ_HANDLER](#)(id) void id(void)
Fast IRQ handler function declaration.
- #define [port_switch](#)(ntp, otp) [_port_switch](#)(ntp, otp)
Performs a context switch between two threads.

Architecture and Compiler

- #define [PORT_ARCHITECTURE_XXX](#)

- `#define PORT_ARCHITECTURE_XXX_YYY`
Macro defining the specific XXX architecture.
- `#define PORT_ARCHITECTURE_NAME "XXX Architecture"`
Name of the implemented architecture.
- `#define PORT_COMPILER_NAME "GCC" __VERSION__`
Compiler name and version.
- `#define PORT_SUPPORTS_RT FALSE`
This port supports a realtime counter.
- `#define PORT_INFO "no info"`
Port-specific information string.

Typedefs

- `typedef uint64_t stkalign_t`
Type of stack and memory alignment enforcement.

Functions

- `void _port_init (void)`
Port-related initialization code.
- `void _port_switch (thread_t *ntp, thread_t *otp)`
Performs a context switch between two threads.
- `static syssts_t port_get_irq_status (void)`
Returns a word encoding the current interrupts status.
- `static bool port_irq_enabled (syssts_t sts)`
Checks the interrupt status.
- `static bool port_is_isr_context (void)`
Determines the current execution context.
- `static void port_lock (void)`
Kernel-lock action.
- `static void port_unlock (void)`
Kernel-unlock action.
- `static void port_lock_from_isr (void)`
Kernel-lock action from an interrupt handler.
- `static void port_unlock_from_isr (void)`
Kernel-unlock action from an interrupt handler.
- `static void port_disable (void)`
Disables all the interrupt sources.
- `static void port_suspend (void)`
Disables the interrupt sources below kernel-level priority.
- `static void port_enable (void)`
Enables all the interrupt sources.
- `static void port_wait_for_interrupt (void)`
Enters an architecture-dependent IRQ-waiting mode.
- `static rtcnt_t port_rt_get_counter_value (void)`
Returns the current value of the realtime counter.

11.7.1 Detailed Description

Port related template macros and structures.

This file is a template of the system driver macros provided by a port.

11.8 chcustomer.h File Reference

Customer-related info.

Macros

- `#define CH_CUSTOMER_ID_STRING "Santa, North Pole"`
Customer readable identifier.
- `#define CH_CUSTOMER_ID_CODE "xxxx-yyyy"`
Customer code.

11.8.1 Detailed Description

Customer-related info.

11.9 chdebug.c File Reference

ChibiOS/RT Debug code.

```
#include "ch.h"
```

Functions

- `void _dbg_check_disable (void)`
Guard code for `chSysDisable ()`.
- `void _dbg_check_suspend (void)`
Guard code for `chSysSuspend ()`.
- `void _dbg_check_enable (void)`
Guard code for `chSysEnable ()`.
- `void _dbg_check_lock (void)`
Guard code for `chSysLock ()`.
- `void _dbg_check_unlock (void)`
Guard code for `chSysUnlock ()`.
- `void _dbg_check_lock_from_isr (void)`
Guard code for `chSysLockFromIsr ()`.
- `void _dbg_check_unlock_from_isr (void)`
Guard code for `chSysUnlockFromIsr ()`.
- `void _dbg_check_enter_isr (void)`
Guard code for `CH_IRQ_PROLOGUE ()`.
- `void _dbg_check_leave_isr (void)`
Guard code for `CH_IRQ_EPILOGUE ()`.
- `void chDbgCheckClassI (void)`
I-class functions context check.
- `void chDbgCheckClassS (void)`
S-class functions context check.
- `void _dbg_trace_init (void)`
Trace circular buffer subsystem initialization.
- `void _dbg_trace (thread_t *otp)`
Inserts in the circular debug trace buffer a context switch record.

11.9.1 Detailed Description

ChibiOS/RT Debug code.

11.10 chdebug.h File Reference

Debug macros and structures.

Data Structures

- struct [ch_swc_event_t](#)
Trace buffer record.
- struct [ch_trace_buffer_t](#)
Trace buffer header.

Macros

Debug related settings

- #define [CH_DBG_TRACE_BUFFER_SIZE](#) 64
Trace buffer entries.
- #define [CH_DBG_STACK_FILL_VALUE](#) 0x55
Fill value for thread stack area in debug mode.
- #define [CH_DBG_THREAD_FILL_VALUE](#) 0xFF
Fill value for thread area in debug mode.

Macro Functions

- #define [chDbgCheck\(c\)](#)
Function parameters check.
- #define [chDbgAssert\(c, r\)](#)
Condition assertion.

Functions

- void [_dbg_trace_init](#) (void)
Trace circular buffer subsystem initialization.
- void [_dbg_trace](#) ([thread_t](#) *otp)
Inserts in the circular debug trace buffer a context switch record.

11.10.1 Detailed Description

Debug macros and structures.

11.11 chdynamic.c File Reference

Dynamic threads code.

```
#include "ch.h"
```

Functions

- `thread_t * chThdAddRef (thread_t *tp)`
Adds a reference to a thread object.
- `void chThdRelease (thread_t *tp)`
Releases a reference to a thread object.
- `thread_t * chThdCreateFromHeap (memory_heap_t *heapp, size_t size, tprio_t prio, tfunc_t pf, void *arg)`
Creates a new thread allocating the memory from the heap.
- `thread_t * chThdCreateFromMemoryPool (memory_pool_t *mp, tprio_t prio, tfunc_t pf, void *arg)`
Creates a new thread allocating the memory from the specified memory pool.

11.11.1 Detailed Description

Dynamic threads code.

11.12 chdynamic.h File Reference

Dynamic threads macros and structures.

Functions

- `thread_t * chThdAddRef (thread_t *tp)`
Adds a reference to a thread object.
- `void chThdRelease (thread_t *tp)`
Releases a reference to a thread object.
- `thread_t * chThdCreateFromHeap (memory_heap_t *heapp, size_t size, tprio_t prio, tfunc_t pf, void *arg)`
Creates a new thread allocating the memory from the heap.
- `thread_t * chThdCreateFromMemoryPool (memory_pool_t *mp, tprio_t prio, tfunc_t pf, void *arg)`
Creates a new thread allocating the memory from the specified memory pool.

11.12.1 Detailed Description

Dynamic threads macros and structures.

11.13 chevents.c File Reference

Events code.

```
#include "ch.h"
```

Functions

- `void chEvtRegisterMaskWithFlags (event_source_t *esp, event_listener_t *elp, eventmask_t events, eventflags_t wflags)`
Registers an Event Listener on an Event Source.
- `void chEvtUnregister (event_source_t *esp, event_listener_t *elp)`
Unregisters an Event Listener from its Event Source.
- `eventmask_t chEvtGetAndClearEvents (eventmask_t events)`

- Clears the pending events specified in the events mask.*
- `eventmask_t chEvtAddEvents (eventmask_t events)`

*Adds (OR) a set of events to the current thread, this is **much** faster than using `chEvtBroadcast ()` or `chEvtSignal ()`.*
 - `void chEvtBroadcastFlagsI (event_source_t *esp, eventflags_t flags)`

Signals all the Event Listeners registered on the specified Event Source.
 - `eventflags_t chEvtGetAndClearFlags (event_listener_t *elp)`

Returns the flags associated to an `event_listener_t`.
 - `void chEvtSignal (thread_t *tp, eventmask_t events)`

Adds a set of event flags directly to the specified `thread_t`.
 - `void chEvtSignalI (thread_t *tp, eventmask_t events)`

Adds a set of event flags directly to the specified `thread_t`.
 - `void chEvtBroadcastFlags (event_source_t *esp, eventflags_t flags)`

Signals all the Event Listeners registered on the specified Event Source.
 - `eventflags_t chEvtGetAndClearFlagsI (event_listener_t *elp)`

Returns the flags associated to an `event_listener_t`.
 - `void chEvtDispatch (const evhandler_t *handlers, eventmask_t events)`

Invokes the event handlers associated to an event flags mask.
 - `eventmask_t chEvtWaitOne (eventmask_t events)`

Waits for exactly one of the specified events.
 - `eventmask_t chEvtWaitAny (eventmask_t events)`

Waits for any of the specified events.
 - `eventmask_t chEvtWaitAll (eventmask_t events)`

Waits for all the specified events.
 - `eventmask_t chEvtWaitOneTimeout (eventmask_t events, systime_t time)`

Waits for exactly one of the specified events.
 - `eventmask_t chEvtWaitAnyTimeout (eventmask_t events, systime_t time)`

Waits for any of the specified events.
 - `eventmask_t chEvtWaitAllTimeout (eventmask_t events, systime_t time)`

Waits for all the specified events.

11.13.1 Detailed Description

Events code.

11.14 chevents.h File Reference

Events macros and structures.

Data Structures

- `struct event_listener`

Event Listener structure.
- `struct event_source`

Event Source structure.

Macros

- `#define ALL_EVENTS ((eventmask_t)-1)`
All events allowed mask.
- `#define EVENT_MASK(eid) ((eventmask_t)1 << (eventmask_t)(eid))`
Returns an event mask from an event identifier.
- `#define _EVENTSOURCE_DATA(name) { (void *)(&name)}`
Data part of a static event source initializer.
- `#define EVENTSOURCE_DECL(name) event_source_t name = _EVENTSOURCE_DATA(name)`
Static event source initializer.

TypeDefs

- `typedef struct event_source event_source_t`
Event Source structure.
- `typedef void(* evhandler_t) (eventid_t id)`
Event Handler callback function.

Functions

- `void chEvtRegisterMaskWithFlags (event_source_t *esp, event_listener_t *elp, eventmask_t events, eventflags_t wflags)`
Registers an Event Listener on an Event Source.
- `void chEvtUnregister (event_source_t *esp, event_listener_t *elp)`
Unregisters an Event Listener from its Event Source.
- `eventmask_t chEvtGetAndClearEvents (eventmask_t events)`
Clears the pending events specified in the events mask.
- `eventmask_t chEvtAddEvents (eventmask_t events)`
*Adds (OR) a set of events to the current thread, this is **much** faster than using `chEvtBroadcast ()` or `chEvtSignal ()`.*
- `eventflags_t chEvtGetAndClearFlags (event_listener_t *elp)`
Returns the flags associated to an event_listener_t.
- `eventflags_t chEvtGetAndClearFlagsI (event_listener_t *elp)`
Returns the flags associated to an event_listener_t.
- `void chEvtSignal (thread_t *tp, eventmask_t events)`
Adds a set of event flags directly to the specified thread_t.
- `void chEvtSignall (thread_t *tp, eventmask_t events)`
Adds a set of event flags directly to the specified thread_t.
- `void chEvtBroadcastFlags (event_source_t *esp, eventflags_t flags)`
Signals all the Event Listeners registered on the specified Event Source.
- `void chEvtBroadcastFlagsI (event_source_t *esp, eventflags_t flags)`
Signals all the Event Listeners registered on the specified Event Source.
- `void chEvtDispatch (const evhandler_t *handlers, eventmask_t events)`
Invokes the event handlers associated to an event flags mask.
- `eventmask_t chEvtWaitOne (eventmask_t events)`
Waits for exactly one of the specified events.
- `eventmask_t chEvtWaitAny (eventmask_t events)`
Waits for any of the specified events.
- `eventmask_t chEvtWaitAll (eventmask_t events)`
Waits for all the specified events.
- `eventmask_t chEvtWaitOneTimeout (eventmask_t events, systime_t time)`

- **eventmask_t chEvtWaitAnyTimeout (eventmask_t events, systime_t time)**
Waits for any of the specified events.
- **eventmask_t chEvtWaitAllTimeout (eventmask_t events, systime_t time)**
Waits for all the specified events.
- **static void chEvtObjectInit (event_source_t *esp)**
Initializes an Event Source.
- **static void chEvtRegisterMask (event_source_t *esp, event_listener_t *elp, eventmask_t events)**
Registers an Event Listener on an Event Source.
- **static void chEvtRegister (event_source_t *esp, event_listener_t *elp, eventid_t event)**
Registers an Event Listener on an Event Source.
- **static bool chEvtIsListeningI (event_source_t *esp)**
Verifies if there is at least one event_listener_t registered.
- **static void chEvtBroadcast (event_source_t *esp)**
Signals all the Event Listeners registered on the specified Event Source.
- **static void chEvtBroadcastI (event_source_t *esp)**
Signals all the Event Listeners registered on the specified Event Source.

11.14.1 Detailed Description

Events macros and structures.

11.15 chheap.c File Reference

Heaps code.

```
#include "ch.h"
```

Functions

- **void _heap_init (void)**
Initializes the default heap.
- **void chHeapObjectInit (memory_heap_t *heapp, void *buf, size_t size)**
Initializes a memory heap from a static memory area.
- **void * chHeapAlloc (memory_heap_t *heapp, size_t size)**
Allocates a block of memory from the heap by using the first-fit algorithm.
- **void chHeapFree (void *p)**
Frees a previously allocated memory block.
- **size_t chHeapStatus (memory_heap_t *heapp, size_t *sizep)**
Reports the heap status.

Variables

- **static memory_heap_t default_heap**
Default heap descriptor.

11.15.1 Detailed Description

Heaps code.

11.16 chheap.h File Reference

Heaps macros and structures.

Data Structures

- union `heap_header`
Memory heap block header.
- struct `memory_heap`
Structure describing a memory heap.

Typedefs

- typedef struct `memory_heap` `memory_heap_t`
Type of a memory heap.

Functions

- void `_heap_init` (void)
Initializes the default heap.
- void `chHeapObjectInit` (`memory_heap_t` *heapp, void *buf, size_t size)
Initializes a memory heap from a static memory area.
- void * `chHeapAlloc` (`memory_heap_t` *heapp, size_t size)
Allocates a block of memory from the heap by using the first-fit algorithm.
- void `chHeapFree` (void *p)
Frees a previously allocated memory block.
- size_t `chHeapStatus` (`memory_heap_t` *heapp, size_t *sizep)
Reports the heap status.

11.16.1 Detailed Description

Heaps macros and structures.

11.17 chlicense.h File Reference

License Module macros and structures.

Macros

- #define `CH_LICENSE` `CH_LICENSE_GPL`
Current license.
- #define `CH_LICENSE_TYPE_STRING` "GNU General Public License 3 (GPL3)"
License identification string.
- #define `CH_LICENSE_ID_STRING` "N/A"
Customer identification string.
- #define `CH_LICENSE_ID_CODE` "N/A"
Customer code.
- #define `CH_LICENSE_MODIFIABLE_CODE` TRUE

- `#define CH_LICENSE_FEATURES CH_FEATURES_FULL`
Code modifiability restrictions.
- `#define CH_LICENSE_MAX_DEPLOY CH_DEPLOY_UNLIMITED`
Code functionality restrictions.
- `#define CH_LICENSE_MAX_DEPLOY CH_DEPLOY_UNLIMITED`
Code deploy restrictions.

Allowed Features Levels

- `#define CH_FEATURES_BASIC 0`
- `#define CH_FEATURES_INTERMEDIATE 1`
- `#define CH_FEATURES_FULL 2`

Deployment Options

- `#define CH_DEPLOY_UNLIMITED -1`
- `#define CH_DEPLOY_NONE 0`

Licensing Options

- `#define CH_LICENSE_GPL 0`
- `#define CH_LICENSE_GPL_EXCEPTION 1`
- `#define CH_LICENSE_COMMERCIAL_FREE 2`
- `#define CH_LICENSE_COMMERCIAL_DEVELOPER 3`
- `#define CH_LICENSE_COMMERCIAL_FULL 4`
- `#define CH_LICENSE_PARTNER 5`

11.17.1 Detailed Description

License Module macros and structures.

11.18 chmboxes.c File Reference

Mailboxes code.

```
#include "ch.h"
```

Functions

- `void chMBOBJECTInit (mailbox_t *mbp, msg_t *buf, cnt_t n)`
Initializes a `mailbox_t` object.
- `void chMBReset (mailbox_t *mbp)`
Resets a `mailbox_t` object.
- `void chMBResetI (mailbox_t *mbp)`
Resets a `mailbox_t` object.
- `msg_t chMBPost (mailbox_t *mbp, msg_t msg, systime_t timeout)`
Posts a message into a mailbox.
- `msg_t chMBPostS (mailbox_t *mbp, msg_t msg, systime_t timeout)`
Posts a message into a mailbox.
- `msg_t chMBPostI (mailbox_t *mbp, msg_t msg)`
Posts a message into a mailbox.
- `msg_t chMBPostAhead (mailbox_t *mbp, msg_t msg, systime_t timeout)`
Posts an high priority message into a mailbox.

- `msg_t chMBPostAheadS (mailbox_t *mbp, msg_t msg, systime_t timeout)`
Posts an high priority message into a mailbox.
- `msg_t chMBPostAheadI (mailbox_t *mbp, msg_t msg)`
Posts an high priority message into a mailbox.
- `msg_t chMBFetch (mailbox_t *mbp, msg_t *msgp, systime_t timeout)`
Retrieves a message from a mailbox.
- `msg_t chMBFetchS (mailbox_t *mbp, msg_t *msgp, systime_t timeout)`
Retrieves a message from a mailbox.
- `msg_t chMBFetchI (mailbox_t *mbp, msg_t *msgp)`
Retrieves a message from a mailbox.

11.18.1 Detailed Description

Mailboxes code.

11.19 chmboxes.h File Reference

Mailboxes macros and structures.

Data Structures

- struct `mailbox_t`
Structure representing a mailbox object.

Macros

- `#define _MAILBOX_DATA(name, buffer, size)`
Data part of a static mailbox initializer.
- `#define MAILBOX_DECL(name, buffer, size) mailbox_t name = _MAILBOX_DATA(name, buffer, size)`
Static mailbox initializer.

Functions

- `void chMBOBJECTInit (mailbox_t *mbp, msg_t *buf, cnt_t n)`
Initializes a `mailbox_t` object.
- `void chMBReset (mailbox_t *mbp)`
Resets a `mailbox_t` object.
- `void chMBResetI (mailbox_t *mbp)`
Resets a `mailbox_t` object.
- `msg_t chMBPost (mailbox_t *mbp, msg_t msg, systime_t timeout)`
Posts a message into a mailbox.
- `msg_t chMBPostS (mailbox_t *mbp, msg_t msg, systime_t timeout)`
Posts a message into a mailbox.
- `msg_t chMBPostI (mailbox_t *mbp, msg_t msg)`
Posts a message into a mailbox.
- `msg_t chMBPostAhead (mailbox_t *mbp, msg_t msg, systime_t timeout)`
Posts an high priority message into a mailbox.
- `msg_t chMBPostAheadS (mailbox_t *mbp, msg_t msg, systime_t timeout)`
Posts an high priority message into a mailbox.

- `msg_t chMBPostAhead1 (mailbox_t *mbp, msg_t msg)`
Posts an high priority message into a mailbox.
- `msg_t chMBFetch (mailbox_t *mbp, msg_t *msgp, systime_t timeout)`
Retrieves a message from a mailbox.
- `msg_t chMBFetchS (mailbox_t *mbp, msg_t *msgp, systime_t timeout)`
Retrieves a message from a mailbox.
- `msg_t chMBFetchl (mailbox_t *mbp, msg_t *msgp)`
Retrieves a message from a mailbox.
- `static size_t chMBGetSizel (mailbox_t *mbp)`
Returns the mailbox buffer size.
- `static cnt_t chMBGetFreeCountl (mailbox_t *mbp)`
Returns the number of free message slots into a mailbox.
- `static cnt_t chMBGetUsedCountl (mailbox_t *mbp)`
Returns the number of used message slots into a mailbox.
- `static msg_t chMBPeekl (mailbox_t *mbp)`
Returns the next message in the queue without removing it.

11.19.1 Detailed Description

Mailboxes macros and structures.

11.20 chmemcore.c File Reference

Core memory manager code.

```
#include "ch.h"
```

Functions

- `void _core_init (void)`
Low level memory manager initialization.
- `void * chCoreAlloc (size_t size)`
Allocates a memory block.
- `void * chCoreAllocl (size_t size)`
Allocates a memory block.
- `size_t chCoreGetStatusX (void)`
Core memory status.

11.20.1 Detailed Description

Core memory manager code.

11.21 chmemcore.h File Reference

Core memory manager macros and structures.

Macros

Alignment support macros

- `#define MEM_ALIGN_SIZE sizeof(stkalign_t)`
Alignment size constant.
- `#define MEM_ALIGN_MASK (MEM_ALIGN_SIZE - 1U)`
Alignment mask constant.
- `#define MEM_ALIGN_PREV(p) ((size_t)(p) & ~MEM_ALIGN_MASK)`
Alignment helper macro.
- `#define MEM_ALIGN_NEXT(p) MEM_ALIGN_PREV((size_t)(p) + MEM_ALIGN_MASK)`
Alignment helper macro.
- `#define MEM_IS_ALIGNED(p) (((size_t)(p) & MEM_ALIGN_MASK) == 0U)`
Returns whatever a pointer or memory size is aligned to the type `stkalign_t`.

Typedefs

- `typedef void *(*memgetfunc_t) (size_t size)`
Memory get function.

Functions

- `void _core_init (void)`
Low level memory manager initialization.
- `void * chCoreAlloc (size_t size)`
Allocates a memory block.
- `void * chCoreAlloc1 (size_t size)`
Allocates a memory block.
- `size_t chCoreGetStatusX (void)`
Core memory status.

11.21.1 Detailed Description

Core memory manager macros and structures.

11.22 chmempools.c File Reference

Memory Pools code.

```
#include "ch.h"
```

Functions

- `void chPoolObjectInit (memory_pool_t *mp, size_t size, memgetfunc_t provider)`
Initializes an empty memory pool.
- `void chPoolLoadArray (memory_pool_t *mp, void *p, size_t n)`
Loads a memory pool with an array of static objects.
- `void * chPoolAlloc1 (memory_pool_t *mp)`
Allocates an object from a memory pool.
- `void * chPoolAlloc (memory_pool_t *mp)`

- `void chPoolFree (memory_pool_t *mp, void *objp)`
Releases an object into a memory pool.
- `void chPoolFree (memory_pool_t *mp, void *objp)`
Releases an object into a memory pool.

11.22.1 Detailed Description

Memory Pools code.

11.23 chmempools.h File Reference

Memory Pools macros and structures.

Data Structures

- `struct pool_header`
Memory pool free object header.
- `struct memory_pool_t`
Memory pool descriptor.

Macros

- `#define _MEMORYPOOL_DATA(name, size, provider) {NULL, size, provider}`
Data part of a static memory pool initializer.
- `#define MEMORYPOOL_DECL(name, size, provider) memory_pool_t name = _MEMORYPOOL_DATA(name, size, provider)`
Static memory pool initializer in hungry mode.

Functions

- `void chPoolObjectInit (memory_pool_t *mp, size_t size, memgetfunc_t provider)`
Initializes an empty memory pool.
- `void chPoolLoadArray (memory_pool_t *mp, void *p, size_t n)`
Loads a memory pool with an array of static objects.
- `void * chPoolAlloc (memory_pool_t *mp)`
Allocates an object from a memory pool.
- `void * chPoolAlloc (memory_pool_t *mp)`
Allocates an object from a memory pool.
- `void chPoolFree (memory_pool_t *mp, void *objp)`
Releases an object into a memory pool.
- `void chPoolFree (memory_pool_t *mp, void *objp)`
Releases an object into a memory pool.
- `static void chPoolAdd (memory_pool_t *mp, void *objp)`
Adds an object to a memory pool.
- `static void chPoolAddl (memory_pool_t *mp, void *objp)`
Adds an object to a memory pool.

11.23.1 Detailed Description

Memory Pools macros and structures.

11.24 chmsg.c File Reference

Messages code.

```
#include "ch.h"
```

Functions

- [msg_t chMsgSend \(thread_t *tp, msg_t msg\)](#)
Sends a message to the specified thread.
- [thread_t * chMsgWait \(void\)](#)
Suspends the thread and waits for an incoming message.
- [void chMsgRelease \(thread_t *tp, msg_t msg\)](#)
Releases a sender thread specifying a response message.

11.24.1 Detailed Description

Messages code.

11.25 chmsg.h File Reference

Messages macros and structures.

Functions

- [msg_t chMsgSend \(thread_t *tp, msg_t msg\)](#)
Sends a message to the specified thread.
- [thread_t * chMsgWait \(void\)](#)
Suspends the thread and waits for an incoming message.
- [void chMsgRelease \(thread_t *tp, msg_t msg\)](#)
Releases a sender thread specifying a response message.
- [static bool chMsgIsPending \(thread_t *tp\)](#)
Evaluates to `true` if the thread has pending messages.
- [static msg_t chMsgGet \(thread_t *tp\)](#)
Returns the message carried by the specified thread.
- [static void chMsgReleaseS \(thread_t *tp, msg_t msg\)](#)
Releases the thread waiting on top of the messages queue.

11.25.1 Detailed Description

Messages macros and structures.

11.26 chmtx.c File Reference

Mutexes code.

```
#include "ch.h"
```

Functions

- void **chMtxObjectInit** (**mutex_t** *mp)
Initializes a mutex_t structure.
- void **chMtxLock** (**mutex_t** *mp)
Locks the specified mutex.
- void **chMtxLockS** (**mutex_t** *mp)
Locks the specified mutex.
- bool **chMtxTryLock** (**mutex_t** *mp)
Tries to lock a mutex.
- bool **chMtxTryLockS** (**mutex_t** *mp)
Tries to lock a mutex.
- void **chMtxUnlock** (**mutex_t** *mp)
Unlocks the specified mutex.
- void **chMtxUnlockS** (**mutex_t** *mp)
Unlocks the specified mutex.
- void **chMtxUnlockAll** (void)
Unlocks all mutexes owned by the invoking thread.

11.26.1 Detailed Description

Mutexes code.

11.27 chmtx.h File Reference

Mutexes macros and structures.

Data Structures

- struct **ch_mutex**
Mutex structure.

Macros

- #define **_MUTEX_DATA**(name) {**_THREADS_QUEUE_DATA**(name.m_queue), NULL, NULL, 0}
Data part of a static mutex initializer.
- #define **MUTEX_DECL**(name) **mutex_t** name = **_MUTEX_DATA**(name)
Static mutex initializer.

Typedefs

- typedef struct **ch_mutex mutex_t**
Type of a mutex structure.

Functions

- void **chMtxObjectInit** (**mutex_t** *mp)

Initializes a mutex structure.
- void **chMtxLock** (**mutex_t** *mp)

Locks the specified mutex.
- void **chMtxLockS** (**mutex_t** *mp)

Locks the specified mutex.
- bool **chMtxTryLock** (**mutex_t** *mp)

Tries to lock a mutex.
- bool **chMtxTryLockS** (**mutex_t** *mp)

Tries to lock a mutex.
- void **chMtxUnlock** (**mutex_t** *mp)

Unlocks the specified mutex.
- void **chMtxUnlockS** (**mutex_t** *mp)

Unlocks the specified mutex.
- void **chMtxUnlockAll** (void)

Unlocks all mutexes owned by the invoking thread.
- static bool **chMtxQueueNotEmptyS** (**mutex_t** *mp)

Returns true if the mutex queue contains at least a waiting thread.
- static **mutex_t** * **chMtxGetNextMutexS** (void)

Returns the next mutex in the mutexes stack of the current thread.

11.27.1 Detailed Description

Mutexes macros and structures.

11.28 chqueues.c File Reference

I/O Queues code.

```
#include "ch.h"
```

Functions

- void **chIQObjectInit** (**input_queue_t** *iwp, **uint8_t** *bp, **size_t** size, **qnotify_t** infy, void *link)

Initializes an input queue.
- void **chIQResetI** (**input_queue_t** *iwp)

Resets an input queue.
- **msg_t** **chIQPutI** (**input_queue_t** *iwp, **uint8_t** b)

Input queue write.
- **msg_t** **chIQGetTimeout** (**input_queue_t** *iwp, **systime_t** timeout)

Input queue read with timeout.
- **size_t** **chIQReadTimeout** (**input_queue_t** *iwp, **uint8_t** *bp, **size_t** n, **systime_t** timeout)

Input queue read with timeout.
- void **chOQObjectInit** (**output_queue_t** *oqp, **uint8_t** *bp, **size_t** size, **qnotify_t** onfy, void *link)

Initializes an output queue.
- void **chOQResetI** (**output_queue_t** *oqp)

Resets an output queue.

- `msg_t chOQPutTimeout (output_queue_t *oqp, uint8_t b, systime_t timeout)`
Output queue write with timeout.
- `msg_t chOQGetI (output_queue_t *oqp)`
Output queue read.
- `size_t chOQWriteTimeout (output_queue_t *oqp, const uint8_t *bp, size_t n, systime_t timeout)`
Output queue write with timeout.

11.28.1 Detailed Description

I/O Queues code.

11.29 chqueues.h File Reference

I/O Queues macros and structures.

Data Structures

- `struct io_queue`
Generic I/O queue structure.

Macros

- `#define _INPUTQUEUE_DATA(name, buffer, size, inotify, link)`
Data part of a static input queue initializer.
- `#define INPUTQUEUE_DECL(name, buffer, size, inotify, link) input_queue_t name = _INPUTQUEUE_DATA(name, buffer, size, inotify, link)`
Static input queue initializer.
- `#define _OUTPUTQUEUE_DATA(name, buffer, size, onotify, link)`
Data part of a static output queue initializer.
- `#define OUTPUTQUEUE_DECL(name, buffer, size, onotify, link) output_queue_t name = _OUTPUTQUEUE_DATA(name, buffer, size, onotify, link)`
Static output queue initializer.

Queue functions returned status value

- `#define Q_OK MSG_OK`
Operation successful.
- `#define Q_TIMEOUT MSG_TIMEOUT`
Timeout condition.
- `#define Q_RESET MSG_RESET`
Queue has been reset.
- `#define Q_EMPTY (msg_t)-3`
Queue empty.
- `#define Q_FULL (msg_t)-4`
Queue full.

Macro Functions

- `#define chQSizeX(qp)`
Returns the queue's buffer size.
- `#define chQSpaceI(qp) ((qp)->q_counter)`
Queue space.
- `#define chQGetLinkX(qp) ((qp)->q_link)`
Returns the queue application-defined link.

Typedefs

- **typedef struct io_queue io_queue_t**
Type of a generic I/O queue structure.
- **typedef void(* qnotify_t) (io_queue_t *qp)**
Queue notification callback type.
- **typedef io_queue_t input_queue_t**
Type of an input queue structure.
- **typedef io_queue_t output_queue_t**
Type of an output queue structure.

Functions

- **void chIQObjectInit (input_queue_t *iqp, uint8_t *bp, size_t size, qnotify_t infy, void *link)**
Initializes an input queue.
- **void chIQResetl (input_queue_t *iqp)**
Resets an input queue.
- **msg_t chIQPutl (input_queue_t *iqp, uint8_t b)**
Input queue write.
- **msg_t chIQGetTimeout (input_queue_t *iqp, systime_t timeout)**
Input queue read with timeout.
- **size_t chIQReadTimeout (input_queue_t *iqp, uint8_t *bp, size_t n, systime_t timeout)**
Input queue read with timeout.
- **void chOQObjectInit (output_queue_t *oqp, uint8_t *bp, size_t size, qnotify_t onfy, void *link)**
Initializes an output queue.
- **void chOQResetl (output_queue_t *oqp)**
Resets an output queue.
- **msg_t chOQPutTimeout (output_queue_t *oqp, uint8_t b, systime_t timeout)**
Output queue write with timeout.
- **msg_t chOQGetl (output_queue_t *oqp)**
Output queue read.
- **size_t chOQWriteTimeout (output_queue_t *oqp, const uint8_t *bp, size_t n, systime_t timeout)**
Output queue write with timeout.
- **static size_t chIQGetFulll (input_queue_t *iqp)**
Returns the filled space into an input queue.
- **static size_t chIQGetEmptyl (input_queue_t *iqp)**
Returns the empty space into an input queue.
- **static bool chIQIsEmptyl (input_queue_t *iqp)**
Evaluates to true if the specified input queue is empty.
- **static bool chIQIsFulll (input_queue_t *iqp)**
Evaluates to true if the specified input queue is full.
- **static msg_t chIQGet (input_queue_t *iqp)**
Input queue read.
- **static size_t chOQGetFulll (output_queue_t *oqp)**
Returns the filled space into an output queue.
- **static size_t chOQGetEmptyl (output_queue_t *oqp)**
Returns the empty space into an output queue.
- **static bool chOQIsEmptyl (output_queue_t *oqp)**
Evaluates to true if the specified output queue is empty.
- **static bool chOQIsFulll (output_queue_t *oqp)**
Evaluates to true if the specified output queue is full.
- **static msg_t chOQPut (output_queue_t *oqp, uint8_t b)**
Output queue write.

11.29.1 Detailed Description

I/O Queues macros and structures.

11.30 chregistry.c File Reference

Threads registry code.

```
#include "ch.h"
```

Functions

- **thread_t * chRegFirstThread (void)**
Returns the first thread in the system.
- **thread_t * chRegNextThread (thread_t *tp)**
Returns the thread next to the specified one.

11.30.1 Detailed Description

Threads registry code.

11.31 chregistry.h File Reference

Threads registry macros and structures.

Data Structures

- struct **chdebug_t**
ChibiOS/RT memory signature record.

Macros

- #define **REG_REMOVE(tp)**
Removes a thread from the registry list.
- #define **REG_INSERT(tp)**
Adds a thread to the registry list.

Functions

- **thread_t * chRegFirstThread (void)**
Returns the first thread in the system.
- **thread_t * chRegNextThread (thread_t *tp)**
Returns the thread next to the specified one.
- static void **chRegSetThreadName (const char *name)**
Sets the current thread name.
- static const char * **chRegGetThreadNameX (thread_t *tp)**
Returns the name of the specified thread.
- static void **chRegSetThreadNameX (thread_t *tp, const char *name)**
Changes the name of the specified thread.

11.31.1 Detailed Description

Threads registry macros and structures.

11.32 chschd.c File Reference

Scheduler code.

```
#include "ch.h"
```

Functions

- **void _scheduler_init (void)**
Scheduler initialization.
- **void queue_prio_insert (thread_t *tp, threads_queue_t *tqp)**
Inserts a thread into a priority ordered queue.
- **void queue_insert (thread_t *tp, threads_queue_t *tqp)**
Inserts a thread into a queue.
- **thread_t * queue_fifo_remove (threads_queue_t *tqp)**
Removes the first-out thread from a queue and returns it.
- **thread_t * queue_lifo_remove (threads_queue_t *tqp)**
Removes the last-out thread from a queue and returns it.
- **thread_t * queue_dequeue (thread_t *tp)**
Removes a thread from a queue and returns it.
- **void list_insert (thread_t *tp, threads_list_t *tlp)**
Pushes a thread_t on top of a stack list.
- **thread_t * list_remove (threads_list_t *tlp)**
Pops a thread from the top of a stack list and returns it.
- **thread_t * chSchReadyI (thread_t *tp)**
Inserts a thread in the Ready List.
- **void chSchGoSleepS (tstate_t newstate)**
Puts the current thread to sleep into the specified state.
- **msg_t chSchGoSleepTimeoutS (tstate_t newstate, systime_t time)**
Puts the current thread to sleep into the specified state with timeout specification.
- **void chSchWakeupS (thread_t *ntp, msg_t msg)**
Wakes up a thread.
- **void chSchRescheduleS (void)**
Performs a reschedule if a higher priority thread is runnable.
- **bool chSchIsPreemptionRequired (void)**
Evaluates if preemption is required.
- **void chSchDoRescheduleBehind (void)**
Switches to the first thread on the runnable queue.
- **void chSchDoRescheduleAhead (void)**
Switches to the first thread on the runnable queue.
- **void chSchDoReschedule (void)**
Switches to the first thread on the runnable queue.

Variables

- `ch_system_t ch`
System data structures.

11.32.1 Detailed Description

Scheduler code.

11.33 chschd.h File Reference

Scheduler macros and structures.

Data Structures

- struct `ch_threads_list`
Generic threads single link list, it works like a stack.
- struct `ch_threads_queue`
Generic threads bidirectional linked list header and element.
- struct `ch_thread`
Structure representing a thread.
- struct `ch_virtual_timer`
Virtual Timer descriptor structure.
- struct `ch_virtual_timers_list`
Virtual timers list header.
- struct `ch_system_debug`
System debug data structure.
- struct `ch_system`
System data structure.

Macros

- #define `firstprio`(rlp) `((rlp)->p_next->p_prio)`
Returns the priority of the first thread on the given ready list.
- #define `currp` `ch.rlist.r_current`
Current thread pointer access macro.
- #define `setcurrp`(tp) `(currp = (tp))`
Current thread pointer change macro.

Wakeup status codes

- #define `MSG_OK` `(msg_t)0`
Normal wakeup message.
- #define `MSG_TIMEOUT` `(msg_t)-1`
Wakeup caused by a timeout condition.
- #define `MSG_RESET` `(msg_t)-2`
Wakeup caused by a reset condition.

Priority constants

- #define `NOPRIO` `(tprio_t)0`

- `#define IDLEPRIO (tprio_t)1`
Idle priority.
- `#define LOWPRIO (tprio_t)2`
Lowest priority.
- `#define NORMALPRIO (tprio_t)64`
Normal priority.
- `#define HIGHPRIO (tprio_t)127`
Highest priority.
- `#define ABSPRIO (tprio_t)255`
Greatest priority.

Thread states

- `#define CH_STATE_READY (tstate_t)0`
Waiting on the ready list.
- `#define CH_STATE_CURRENT (tstate_t)1`
Currently running.
- `#define CH_STATE_WTSTART (tstate_t)2`
Just created.
- `#define CH_STATE_SUSPENDED (tstate_t)3`
Suspended state.
- `#define CH_STATE_QUEUED (tstate_t)4`
On an I/O queue.
- `#define CH_STATE_WTSEM (tstate_t)5`
On a semaphore.
- `#define CH_STATE_WTMutex (tstate_t)6`
On a mutex.
- `#define CH_STATE_WTCOND (tstate_t)7`
On a cond.variable.
- `#define CH_STATE_SLEEPING (tstate_t)8`
Sleeping.
- `#define CH_STATE_WTEXIT (tstate_t)9`
Waiting a thread.
- `#define CH_STATE_WTOREVT (tstate_t)10`
One event.
- `#define CH_STATE_WTANDEVT (tstate_t)11`
Several events.
- `#define CH_STATE SNDMSGQ (tstate_t)12`
Sending a message, in queue.
- `#define CH_STATE SNDMSG (tstate_t)13`
Sent a message, waiting answer.
- `#define CH_STATE_WTMSG (tstate_t)14`
Waiting for a message.
- `#define CH_STATE_FINAL (tstate_t)15`
Thread terminated.
- `#define CH_STATE_NAMES`
Thread states as array of strings.

Thread flags and attributes

- `#define CH_FLAG_MODE_MASK (tmode_t)3U`
Thread memory mode mask.
- `#define CH_FLAG_MODE_STATIC (tmode_t)0U`
Static thread.
- `#define CH_FLAG_MODE_HEAP (tmode_t)1U`
Thread allocated from a Memory Heap.
- `#define CH_FLAG_MODE_MPOOL (tmode_t)2U`
Thread allocated from a Memory Pool.

- `#define CH_FLAG_TERMINATE (tmode_t)4U`
Termination requested flag.

Working Areas and Alignment

- `#define THD_ALIGN_STACK_SIZE(n) (((((size_t)(n)) - 1U) | (sizeof(stkalign_t) - 1U)) + 1U)`
Enforces a correct alignment for a stack area size value.
- `#define THD_WORKING_AREA_SIZE(n) THD_ALIGN_STACK_SIZE(sizeof(thread_t) + PORT_WA_SIZE(n))`
Calculates the total Working Area size.
- `#define THD_WORKING_AREA(s, n) stkalign_t s[THD_WORKING_AREA_SIZE(n) / sizeof(stkalign_t)]`
Static working area allocation.

Threads abstraction macros

- `#define THD_FUNCTION(tname, arg) PORT_THD_FUNCTION(tname, arg)`
Thread declaration macro.

Functions

- `void _scheduler_init (void)`
Scheduler initialization.
- `thread_t * chSchReadyL (thread_t *tp)`
Inserts a thread in the Ready List.
- `void chSchGoSleepS (tstate_t newstate)`
Puts the current thread to sleep into the specified state.
- `msg_t chSchGoSleepTimeoutS (tstate_t newstate, systime_t time)`
Puts the current thread to sleep into the specified state with timeout specification.
- `void chSchWakeupS (thread_t *ntp, msg_t msg)`
Wakes up a thread.
- `void chSchRescheduleS (void)`
Performs a reschedule if a higher priority thread is runnable.
- `bool chSchIsPreemptionRequired (void)`
Evaluates if preemption is required.
- `void chSchDoRescheduleBehind (void)`
Switches to the first thread on the runnable queue.
- `void chSchDoRescheduleAhead (void)`
Switches to the first thread on the runnable queue.
- `void chSchDoReschedule (void)`
Switches to the first thread on the runnable queue.
- `void queue_prio_insert (thread_t *tp, threads_queue_t *tqp)`
Inserts a thread into a priority ordered queue.
- `void queue_insert (thread_t *tp, threads_queue_t *tqp)`
Inserts a thread into a queue.
- `thread_t * queue_fifo_remove (threads_queue_t *tqp)`
Removes the first-out thread from a queue and returns it.
- `thread_t * queue_lifo_remove (threads_queue_t *tqp)`
Removes the last-out thread from a queue and returns it.
- `thread_t * queue_dequeue (thread_t *tp)`
Removes a thread from a queue and returns it.
- `void list_insert (thread_t *tp, threads_list_t *tlp)`
Pushes a thread_t on top of a stack list.

- **thread_t * list_remove (threads_list_t *tlp)**
Pops a thread from the top of a stack list and returns it.
- static void **list_init (threads_list_t *tlp)**
Threads list initialization.
- static bool **list_isempty (threads_list_t *tlp)**
Evaluates to true if the specified threads list is empty.
- static bool **list_notempty (threads_list_t *tlp)**
Evaluates to true if the specified threads list is not empty.
- static void **queue_init (threads_queue_t *tqp)**
Threads queue initialization.
- static bool **queue_isempty (const threads_queue_t *tqp)**
Evaluates to true if the specified threads queue is empty.
- static bool **queue_notempty (const threads_queue_t *tqp)**
Evaluates to true if the specified threads queue is not empty.
- static bool **chSchIsRescRequired ()** (void)
Determines if the current thread must reschedule.
- static bool **chSchCanYieldS ()** (void)
Determines if yielding is possible.
- static void **chSchDoYieldS ()** (void)
Yields the time slot.
- static void **chSchPreemption ()** (void)
Inline-able preemption code.

11.33.1 Detailed Description

Scheduler macros and structures.

11.34 chsem.c File Reference

Semaphores code.

```
#include "ch.h"
```

Functions

- void **chSemObjectInit (semaphore_t *sp, cnt_t n)**
Initializes a semaphore with the specified counter value.
- void **chSemReset (semaphore_t *sp, cnt_t n)**
Performs a reset operation on the semaphore.
- void **chSemResetI (semaphore_t *sp, cnt_t n)**
Performs a reset operation on the semaphore.
- msg_t **chSemWait (semaphore_t *sp)**
Performs a wait operation on a semaphore.
- msg_t **chSemWaitS (semaphore_t *sp)**
Performs a wait operation on a semaphore.
- msg_t **chSemWaitTimeout (semaphore_t *sp, systime_t time)**
Performs a wait operation on a semaphore with timeout specification.
- msg_t **chSemWaitTimeoutS (semaphore_t *sp, systime_t time)**
Performs a wait operation on a semaphore with timeout specification.

- void `chSemSignal (semaphore_t *sp)`
Performs a signal operation on a semaphore.
- void `chSemSignall (semaphore_t *sp)`
Performs a signal operation on a semaphore.
- void `chSemAddCounterl (semaphore_t *sp, cnt_t n)`
Adds the specified value to the semaphore counter.
- msg_t `chSemSignalWait (semaphore_t *sps, semaphore_t *spw)`
Performs atomic signal and wait operations on two semaphores.

11.34.1 Detailed Description

Semaphores code.

11.35 chsem.h File Reference

Semaphores macros and structures.

Data Structures

- struct `ch_semaphore`
Semaphore structure.

Macros

- #define `_SEMAPHORE_DATA(name, n) {_THREADS_QUEUE_DATA(name.s_queue), n}`
Data part of a static semaphore initializer.
- #define `SEMAPHORE_DECL(name, n) semaphore_t name = _SEMAPHORE_DATA(name, n)`
Static semaphore initializer.

Typedefs

- typedef struct `ch_semaphore` `semaphore_t`
Semaphore structure.

Functions

- void `chSemObjectInit (semaphore_t *sp, cnt_t n)`
Initializes a semaphore with the specified counter value.
- void `chSemReset (semaphore_t *sp, cnt_t n)`
Performs a reset operation on the semaphore.
- void `chSemResetl (semaphore_t *sp, cnt_t n)`
Performs a reset operation on the semaphore.
- msg_t `chSemWait (semaphore_t *sp)`
Performs a wait operation on a semaphore.
- msg_t `chSemWaitS (semaphore_t *sp)`
Performs a wait operation on a semaphore.
- msg_t `chSemWaitTimeout (semaphore_t *sp, systime_t time)`
Performs a wait operation on a semaphore with timeout specification.

- `msg_t chSemWaitTimeoutS (semaphore_t *sp, systime_t time)`
Performs a wait operation on a semaphore with timeout specification.
- `void chSemSignal (semaphore_t *sp)`
Performs a signal operation on a semaphore.
- `void chSemSignall (semaphore_t *sp)`
Performs a signal operation on a semaphore.
- `void chSemAddCounterI (semaphore_t *sp, cnt_t n)`
Adds the specified value to the semaphore counter.
- `msg_t chSemSignalWait (semaphore_t *sp, semaphore_t *spw)`
Performs atomic signal and wait operations on two semaphores.
- `static void chSemFastWaitI (semaphore_t *sp)`
Decreases the semaphore counter.
- `static void chSemFastSignall (semaphore_t *sp)`
Increases the semaphore counter.
- `static cnt_t chSemGetCounterI (semaphore_t *sp)`
Returns the semaphore counter current value.

11.35.1 Detailed Description

Semaphores macros and structures.

11.36 chstats.c File Reference

Statistics module code.

```
#include "ch.h"
```

Functions

- `void _stats_init (void)`
Initializes the statistics module.
- `void _stats_increase_irq (void)`
Increases the IRQ counter.
- `void _stats_ctxswc (thread_t *ntp, thread_t *otp)`
Updates context switch related statistics.
- `void _stats_start_measure_crit_thd (void)`
Starts the measurement of a thread critical zone.
- `void _stats_stop_measure_crit_thd (void)`
Stops the measurement of a thread critical zone.
- `void _stats_start_measure_crit_isr (void)`
Starts the measurement of an ISR critical zone.
- `void _stats_stop_measure_crit_isr (void)`
Stops the measurement of an ISR critical zone.

11.36.1 Detailed Description

Statistics module code.

11.37 chstats.h File Reference

Statistics module macros and structures.

Data Structures

- struct `kernel_stats_t`
Type of a kernel statistics structure.

Functions

- void `_stats_init` (void)
Initializes the statistics module.
- void `_stats_increase_irq` (void)
Increases the IRQ counter.
- void `_stats_ctxswc` (`thread_t *ntp, thread_t *otp`)
Updates context switch related statistics.
- void `_stats_start_measure_crit_thd` (void)
Starts the measurement of a thread critical zone.
- void `_stats_stop_measure_crit_thd` (void)
Stops the measurement of a thread critical zone.
- void `_stats_start_measure_crit_isr` (void)
Starts the measurement of an ISR critical zone.
- void `_stats_stop_measure_crit_isr` (void)
Stops the measurement of an ISR critical zone.

11.37.1 Detailed Description

Statistics module macros and structures.

11.38 chstreams.h File Reference

Data streams.

Data Structures

- struct `BaseSequentialStreamVMT`
BaseSequentialStream virtual methods table.
- struct `BaseSequentialStream`
Base stream class.

Macros

- `#define _base_sequential_stream_methods`
BaseSequentialStream specific methods.
- `#define _base_sequential_stream_data`
BaseSequentialStream specific data.

Macro Functions (BaseSequentialStream)

- #define **chSequentialStreamWrite**(ip, bp, n) ((ip)->vmt->write(ip, bp, n))
Sequential Stream write.
- #define **chSequentialStreamRead**(ip, bp, n) ((ip)->vmt->read(ip, bp, n))
Sequential Stream read.
- #define **chSequentialStreamPut**(ip, b) ((ip)->vmt->put(ip, b))
Sequential Stream blocking byte write.
- #define **chSequentialStreamGet**(ip) ((ip)->vmt->get(ip))
Sequential Stream blocking byte read.

11.38.1 Detailed Description

Data streams.

This header defines abstract interfaces useful to access generic data streams in a standardized way.

11.39 chsys.c File Reference

System related code.

```
#include "ch.h"
```

Functions

- static void **_idle_thread** (void *p)
This function implements the idle thread infinite loop.
- void **chSysInit** (void)
ChibiOS/RT initialization.
- void **chSysHalt** (const char *reason)
Halts the system.
- bool **chSysIntegrityCheck** (unsigned testmask)
System integrity check.
- void **chSysTimerHandler** (void)
Handles time ticks for round robin preemption and timer increments.
- **syssts_t** **chSysGetStatusAndLockX** (void)
Returns the execution status and enters a critical zone.
- void **chSysRestoreStatusX** (**syssts_t** sts)
Restores the specified execution status and leaves a critical zone.
- bool **chSysIsCounterWithinX** (**rtcnt_t** cnt, **rtcnt_t** start, **rtcnt_t** end)
Realtime window test.
- void **chSysPolledDelayX** (**rtcnt_t** cycles)
Polled delay.

11.39.1 Detailed Description

System related code.

11.40 chsys.h File Reference

System related macros and structures.

Macros

- `#define chSysGetRealtimeCounterX() (rtcnt_t)port_rt_get_counter_value()`
Returns the current value of the system real time counter.
- `#define chSysSwitch(ntp, otp)`
Performs a context switch.

Masks of executable integrity checks.

- `#define CH_INTEGRITY_RLIST 1U`
- `#define CH_INTEGRITY_VTLIST 2U`
- `#define CH_INTEGRITY_REGISTRY 4U`
- `#define CH_INTEGRITY_PORT 8U`

ISRs abstraction macros

- `#define CH_IRQ_IS_VALID_PRIORITY(prio) PORT_IRQ_IS_VALID_PRIORITY(prio)`
Priority level validation macro.
- `#define CH_IRQ_IS_VALID_KERNEL_PRIORITY(prio) PORT_IRQ_IS_VALID_KERNEL_PRIORITY(prio)`
Priority level validation macro.
- `#define CH_IRQ_PROLOGUE()`
IRQ handler enter code.
- `#define CH_IRQ_EPILOGUE()`
IRQ handler exit code.
- `#define CH_IRQ_HANDLER(id) PORT_IRQ_HANDLER(id)`
Standard normal IRQ handler declaration.

Fast ISRs abstraction macros

- `#define CH_FAST_IRQ_HANDLER(id) PORT_FAST_IRQ_HANDLER(id)`
Standard fast IRQ handler declaration.

Time conversion utilities for the realtime counter

- `#define S2RTC(freq, sec) ((freq) * (sec))`
Seconds to realtime counter.
- `#define MS2RTC(freq, msec) (rtcnt_t)((((freq) + 999UL) / 1000UL) * (msec))`
Milliseconds to realtime counter.
- `#define US2RTC(freq, usec) (rtcnt_t)((((freq) + 999999UL) / 1000000UL) * (usec))`
Microseconds to realtime counter.
- `#define RTC2S(freq, n) (((n) - 1UL) / (freq)) + 1UL`
Realtime counter cycles to seconds.
- `#define RTC2MS(freq, n) (((((n) - 1UL) / ((freq) / 1000UL)) + 1UL)`
Realtime counter cycles to milliseconds.
- `#define RTC2US(freq, n) (((((n) - 1UL) / ((freq) / 1000000UL)) + 1UL)`
Realtime counter cycles to microseconds.

Functions

- `void chSysInit (void)`
ChibiOS/RT initialization.
- `void chSysHalt (const char *reason)`
Halts the system.
- `bool chSysIntegrityCheck1 (unsigned testmask)`
System integrity check.

- void [chSysTimerHandlerl](#) (void)
Handles time ticks for round robin preemption and timer increments.
- [syssts_t chSysGetStatusAndLockX](#) (void)
Returns the execution status and enters a critical zone.
- void [chSysRestoreStatusX](#) ([syssts_t](#) sts)
Restores the specified execution status and leaves a critical zone.
- static void [chSysDisable](#) (void)
Raises the system interrupt priority mask to the maximum level.
- static void [chSysSuspend](#) (void)
Raises the system interrupt priority mask to system level.
- static void [chSysEnable](#) (void)
Lowers the system interrupt priority mask to user level.
- static void [chSysLock](#) (void)
Enters the kernel lock state.
- static void [chSysUnlock](#) (void)
Leaves the kernel lock state.
- static void [chSysLockFromISR](#) (void)
Enters the kernel lock state from within an interrupt handler.
- static void [chSysUnlockFromISR](#) (void)
Leaves the kernel lock state from within an interrupt handler.
- static void [chSysUnconditionalLock](#) (void)
Unconditionally enters the kernel lock state.
- static void [chSysUnconditionalUnlock](#) (void)
Unconditionally leaves the kernel lock state.
- static [thread_t](#) * [chSysGetIdleThreadX](#) (void)
Returns a pointer to the idle thread.

11.40.1 Detailed Description

System related macros and structures.

11.41 chsystypes.h File Reference

System types header.

Typedefs

- [typedef uint32_t systime_t](#)
Type of system time.
- [typedef struct ch_thread thread_t](#)
Type of a thread structure.
- [typedef thread_t * thread_reference_t](#)
Type of a thread reference.
- [typedef struct ch_threads_list threads_list_t](#)
Type of a generic threads single link list, it works like a stack.
- [typedef struct ch_threads_queue threads_queue_t](#)
Type of a generic threads bidirectional linked list header and element.
- [typedef struct ch_ready_list ready_list_t](#)
Type of a ready list header.

- **typedef void(* vfunc_t)** (void *p)
Type of a Virtual Timer callback function.
- **typedef struct ch_virtual_timer virtual_timer_t**
Type of a Virtual Timer structure.
- **typedef struct ch_virtual_timers_list virtual_timers_list_t**
Type of virtual timers list header.
- **typedef struct ch_system_debug system_debug_t**
Type of a system debug structure.
- **typedef struct ch_system ch_system_t**
Type of system data structure.

11.41.1 Detailed Description

System types header.

11.42 chthreads.c File Reference

Threads code.

```
#include "ch.h"
```

Functions

- **thread_t * _thread_init (thread_t *tp, tprio_t prio)**
Initializes a thread structure.
- **void _thread_memfill (uint8_t *startp, uint8_t *endp, uint8_t v)**
Memory fill utility.
- **thread_t * chThdCreateI (void *wsp, size_t size, tprio_t prio, tfunc_t pf, void *arg)**
Creates a new thread into a static memory area.
- **thread_t * chThdCreateStatic (void *wsp, size_t size, tprio_t prio, tfunc_t pf, void *arg)**
Creates a new thread into a static memory area.
- **thread_t * chThdStart (thread_t *tp)**
Resumes a thread created with `chThdCreateI()`.
- **tprio_t chThdSetPriority (tprio_t newprio)**
Changes the running thread priority level then reschedules if necessary.
- **void chThdTerminate (thread_t *tp)**
Requests a thread termination.
- **void chThdSleep (systime_t time)**
Suspends the invoking thread for the specified time.
- **void chThdSleepUntil (systime_t time)**
Suspends the invoking thread until the system time arrives to the specified value.
- **systime_t chThdSleepUntilWindowed (systime_t prev, systime_t next)**
Suspends the invoking thread until the system time arrives to the specified value.
- **void chThdYield (void)**
Yields the time slot.
- **void chThdExit (msg_t msg)**
Terminates the current thread.
- **void chThdExitS (msg_t msg)**
Terminates the current thread.

- `msg_t chThdWait (thread_t *tp)`
Blocks the execution of the invoking thread until the specified thread terminates then the exit code is returned.
- `msg_t chThdSuspendS (thread_reference_t *trp)`
Sends the current thread sleeping and sets a reference variable.
- `msg_t chThdSuspendTimeoutS (thread_reference_t *trp, systime_t timeout)`
Sends the current thread sleeping and sets a reference variable.
- `void chThdResumel (thread_reference_t *trp, msg_t msg)`
Wakes up a thread waiting on a thread reference object.
- `void chThdResumeS (thread_reference_t *trp, msg_t msg)`
Wakes up a thread waiting on a thread reference object.
- `void chThdResume (thread_reference_t *trp, msg_t msg)`
Wakes up a thread waiting on a thread reference object.
- `msg_t chThdEnqueueTimeoutS (threads_queue_t *tqp, systime_t timeout)`
Enqueues the caller thread on a threads queue object.
- `void chThdDequeueNextl (threads_queue_t *tqp, msg_t msg)`
Dequeues and wakes up one thread from the threads queue object, if any.
- `void chThdDequeueAlll (threads_queue_t *tqp, msg_t msg)`
Dequeues and wakes up all threads from the threads queue object.

11.42.1 Detailed Description

Threads code.

11.43 chthreads.h File Reference

Threads module macros and structures.

Macros

Threads queues

- `#define _THREADS_QUEUE_DATA(name) {(<thread_t *>)&name, (<thread_t *>)&name}`
Data part of a static threads queue object initializer.
- `#define _THREADS_QUEUE_DECL(name) threads_queue_t name = _THREADS_QUEUE_DATA(name)`
Static threads queue object initializer.

Macro Functions

- `#define chThdSleepSeconds(sec) chThdSleep(S2ST(sec))`
Delays the invoking thread for the specified number of seconds.
- `#define chThdSleepMilliseconds(msec) chThdSleep(MS2ST(msec))`
Delays the invoking thread for the specified number of milliseconds.
- `#define chThdSleepMicroseconds(usec) chThdSleep(US2ST(usec))`
Delays the invoking thread for the specified number of microseconds.

Typedefs

- `typedef void(* tfunc_t) (void *p)`
Thread function.

Functions

- `thread_t * _thread_init (thread_t *tp, tprio_t prio)`
Initializes a thread structure.
- `void _thread_memfill (uint8_t *startp, uint8_t *endp, uint8_t v)`
Memory fill utility.
- `thread_t * chThdCreateI (void *wsp, size_t size, tprio_t prio, tfunc_t pf, void *arg)`
Creates a new thread into a static memory area.
- `thread_t * chThdCreateStatic (void *wsp, size_t size, tprio_t prio, tfunc_t pf, void *arg)`
Creates a new thread into a static memory area.
- `thread_t * chThdStart (thread_t *tp)`
Resumes a thread created with `chThdCreateI ()`.
- `tprio_t chThdSetPriority (tprio_t newprio)`
Changes the running thread priority level then reschedules if necessary.
- `msg_t chThdSuspendS (thread_reference_t *trp)`
Sends the current thread sleeping and sets a reference variable.
- `msg_t chThdSuspendTimeoutS (thread_reference_t *trp, systime_t timeout)`
Sends the current thread sleeping and sets a reference variable.
- `void chThdResumel (thread_reference_t *trp, msg_t msg)`
Wakes up a thread waiting on a thread reference object.
- `void chThdResumeS (thread_reference_t *trp, msg_t msg)`
Wakes up a thread waiting on a thread reference object.
- `void chThdResume (thread_reference_t *trp, msg_t msg)`
Wakes up a thread waiting on a thread reference object.
- `msg_t chThdEnqueueTimeoutS (threads_queue_t *tqp, systime_t timeout)`
Enqueues the caller thread on a threads queue object.
- `void chThdDequeueNextI (threads_queue_t *tqp, msg_t msg)`
Dequeues and wakes up one thread from the threads queue object, if any.
- `void chThdDequeueAllI (threads_queue_t *tqp, msg_t msg)`
Dequeues and wakes up all threads from the threads queue object.
- `void chThdTerminate (thread_t *tp)`
Requests a thread termination.
- `void chThdSleep (systime_t time)`
Suspends the invoking thread for the specified time.
- `void chThdSleepUntil (systime_t time)`
Suspends the invoking thread until the system time arrives to the specified value.
- `systime_t chThdSleepUntilWindowed (systime_t prev, systime_t next)`
Suspends the invoking thread until the system time arrives to the specified value.
- `void chThdYield (void)`
Yields the time slot.
- `void chThdExit (msg_t msg)`
Terminates the current thread.
- `void chThdExitS (msg_t msg)`
Terminates the current thread.
- `msg_t chThdWait (thread_t *tp)`
Blocks the execution of the invoking thread until the specified thread terminates then the exit code is returned.
- `static thread_t * chThdGetSelfX (void)`
Returns a pointer to the current `thread_t`.
- `static tprio_t chThdGetPriorityX (void)`
Returns the current thread priority.
- `static systime_t chThdGetTicksX (thread_t *tp)`

- static bool `chThdTerminatedX` (`thread_t` *`tp`)

Verifies if the specified thread is in the CH_STATE_FINAL state.
- static bool `chThdShouldTerminateX` (`void`)

Verifies if the current thread has a termination request pending.
- static `thread_t` * `chThdStartI` (`thread_t` *`tp`)

Resumes a thread created with `chThdCreateI()`.
- static void `chThdSleepS` (`systime_t` `time`)

Suspends the invoking thread for the specified time.
- static void `chThdQueueObjectInit` (`threads_queue_t` *`tqp`)

Initializes a threads queue object.
- static bool `chThdQueueIsEmpty` (`threads_queue_t` *`tqp`)

Evaluates to true if the specified queue is empty.
- static void `chThdDoDequeueNextI` (`threads_queue_t` *`tqp`, `msg_t` `msg`)

Dequeues and wakes up one thread from the threads queue object.

11.43.1 Detailed Description

Threads module macros and structures.

11.44 chtm.c File Reference

Time Measurement module code.

```
#include "ch.h"
```

Functions

- void `_tm_init` (`void`)

Initializes the time measurement unit.
- void `chTMObjectInit` (`time_measurement_t` *`tmp`)

Initializes a TimeMeasurement object.
- `NOINLINE` void `chTMStartMeasurementX` (`time_measurement_t` *`tmp`)

Starts a measurement.
- `NOINLINE` void `chTMStopMeasurementX` (`time_measurement_t` *`tmp`)

Stops a measurement.
- `NOINLINE` void `chTMChainMeasurementToX` (`time_measurement_t` *`tmp1`, `time_measurement_t` *`tmp2`)

Stops a measurement and chains to the next one using the same time stamp.

11.44.1 Detailed Description

Time Measurement module code.

11.45 chtm.h File Reference

Time Measurement module macros and structures.

Data Structures

- struct `tm_calibration_t`
Type of a time measurement calibration data.
- struct `time_measurement_t`
Type of a Time Measurement object.

Functions

- void `_tm_init` (void)
Initializes the time measurement unit.
- void `chTMOBJECTINIT` (`time_measurement_t` *tmp)
Initializes a TimeMeasurement object.
- `NOINLINE` void `chTMStartMeasurementX` (`time_measurement_t` *tmp)
Starts a measurement.
- `NOINLINE` void `chTMStopMeasurementX` (`time_measurement_t` *tmp)
Stops a measurement.
- `NOINLINE` void `chTMChainMeasurementToX` (`time_measurement_t` *tmp1, `time_measurement_t` *tmp2)
Stops a measurement and chains to the next one using the same time stamp.

11.45.1 Detailed Description

Time Measurement module macros and structures.

11.46 chtypes.h File Reference

System types template.

```
#include <stddef.h>
#include <stdint.h>
#include <stdbool.h>
```

Macros

- `#define ROMCONST const`
ROM constant modifier.
- `#define NOINLINE __attribute__((noinline))`
Makes functions not inlineable.
- `#define PORT_THD_FUNCTION(tname, arg) msg_t tname(void *arg)`
Optimized thread function declaration macro.
- `#define PACKED_VAR __attribute__((packed))`
Packed variable specifier.

Common constants

- `#define FALSE 0`
Generic 'false' boolean constant.
- `#define TRUE (!FALSE)`
Generic 'true' boolean constant.

Typedefs

Kernel types

- `typedef uint32_t rtcnt_t`
- `typedef uint64_t rtttime_t`
- `typedef uint32_t syssts_t`
- `typedef uint8_t tmode_t`
- `typedef uint8_t tstate_t`
- `typedef uint8_t trefs_t`
- `typedef uint8_t tslices_t`
- `typedef uint32_t tprio_t`
- `typedef int32_t msg_t`
- `typedef int32_t eventid_t`
- `typedef uint32_t eventmask_t`
- `typedef uint32_t eventflags_t`
- `typedef int32_t cnt_t`
- `typedef uint32_t ucnt_t`

11.46.1 Detailed Description

System types template.

The types defined in this file may change depending on the target architecture. You may also try to optimize the size of the various types in order to privilege size or performance, be careful in doing so.

11.47 chvt.c File Reference

Time and Virtual Timers module code.

```
#include "ch.h"
```

Functions

- `void _vt_init (void)`
Virtual Timers initialization.
- `void chVTDoSetl (virtual_timer_t *vtp, systime_t delay, vfunc_t vfunc, void *par)`
Enables a virtual timer.
- `void chVTDoResetl (virtual_timer_t *vtp)`
Disables a Virtual Timer.

11.47.1 Detailed Description

Time and Virtual Timers module code.

11.48 chvt.h File Reference

Time and Virtual Timers module macros and structures.

Macros

Special time constants

- #define **TIME_IMMEDIATE** ((**systime_t**)0)
Zero time specification for some functions with a timeout specification.
- #define **TIME_INFINITE** ((**systime_t**)-1)
Infinite time specification for all functions with a timeout specification.

Time conversion utilities

- #define **S2ST(sec)** ((**systime_t**)((**uint32_t**)(sec) * (**uint32_t**)**CH_CFG_ST_FREQUENCY**))
Seconds to system ticks.
- #define **MS2ST(msec)**
Milliseconds to system ticks.
- #define **US2ST(usec)**
Microseconds to system ticks.
- #define **ST2S(n)** (((n) + **CH_CFG_ST_FREQUENCY** - 1UL) / **CH_CFG_ST_FREQUENCY**)
System ticks to seconds.
- #define **ST2MS(n)**
System ticks to milliseconds.
- #define **ST2US(n)**
System ticks to microseconds.

Functions

- void **_vt_init** (void)
Virtual Timers initialization.
- void **chVTDoSetl** (**virtual_timer_t** *vtp, **systime_t** delay, **vfunc_t** vfunc, void *par)
Enables a virtual timer.
- void **chVTDoResetl** (**virtual_timer_t** *vtp)
Disables a Virtual Timer.
- static void **chVTOBJECTInit** (**virtual_timer_t** *vtp)
Initializes a virtual_timer_t object.
- static **systime_t** **chVTGetSystemTimeX** (void)
Current system time.
- static **systime_t** **chVTGetSystemTime** (void)
Current system time.
- static **systime_t** **chVTTIMEElapsedSinceX** (**systime_t** start)
Returns the elapsed time since the specified start time.
- static bool **chVTIsTimeWithinX** (**systime_t** time, **systime_t** start, **systime_t** end)
Checks if the specified time is within the specified time window.
- static bool **chVTIsSystemTimeWithinX** (**systime_t** start, **systime_t** end)
Checks if the current system time is within the specified time window.
- static bool **chVTIsSystemTimeWithin** (**systime_t** start, **systime_t** end)
Checks if the current system time is within the specified time window.
- static bool **chVTGetTimersStateI** (**systime_t** *timep)
Returns the time interval until the next timer event.
- static bool **chVTIsArmedI** (**virtual_timer_t** *vtp)
Returns true if the specified timer is armed.
- static bool **chVTIsArmed** (**virtual_timer_t** *vtp)
Returns true if the specified timer is armed.
- static void **chVTResetl** (**virtual_timer_t** *vtp)
Disables a Virtual Timer.

- static void **chVTReset** (*virtual_timer_t* *vtp)
Disables a Virtual Timer.
- static void **chVTSet1** (*virtual_timer_t* *vtp, *systime_t* delay, *vfunc_t* vfunc, *void* *par)
Enables a virtual timer.
- static void **chVTSet** (*virtual_timer_t* *vtp, *systime_t* delay, *vfunc_t* vfunc, *void* *par)
Enables a virtual timer.
- static void **chVTDotickl** (*void*)
Virtual timers ticker.

11.48.1 Detailed Description

Time and Virtual Timers module macros and structures.

11.49 test.c File Reference

Tests support code.

```
#include "ch.h"
#include "hal.h"
#include "test.h"
#include "testsyst.h"
#include "testthd.h"
#include "testsem.h"
#include "testmtx.h"
#include "testmsg.h"
#include "testmbox.h"
#include "testevt.h"
#include "testheap.h"
#include "testpools.h"
#include "testdyn.h"
#include "testqueues.h"
#include "testbmk.h"
```

Functions

- void **test_printf** (*uint32_t* n)
Prints a decimal unsigned number.
- void **test_puts** (*const char* *msgp)
Prints a line without final end-of-line.
- void **test_printfn** (*const char* *msgp)
Prints a line.
- void **test_emit_token** (*char* token)
Emits a token into the tokens buffer.
- void **test_terminate_threads** (*void*)
Sets a termination request in all the test-spawned threads.
- void **test_wait_threads** (*void*)
Waits for the completion of all the test-spawned threads.
- *systime_t* **test_wait_tick** (*void*)
Delays execution until next system time tick.
- void **test_start_timer** (*unsigned* ms)
Starts the test timer.

- void **TestThread** (void *p)
Test execution thread function.

Variables

- bool **test_timer_done**
Set to TRUE when the test timer reaches its deadline.

11.49.1 Detailed Description

Tests support code.

11.50 test.h File Reference

Tests support header.

Data Structures

- struct **testcase**
Structure representing a test case.

Macros

- #define **DELAY_BETWEEN_TESTS** 200
Delay inserted between test cases.
- #define **TEST_NO_BENCHMARKS** FALSE
If TRUE then benchmarks are not included.
- #define **test_fail**(point)
Test failure enforcement.
- #define **test_assert**(point, condition, msg)
Test assertion.
- #define **test_assert_lock**(point, condition, msg)
Test assertion with lock.
- #define **test_assert_sequence**(point, expected)
Test sequence assertion.
- #define **test_assert_time_window**(point, start, end)
Test time window assertion.

Functions

- void **TestThread** (void *p)
Test execution thread function.
- void **test_printf** (uint32_t n)
Prints a decimal unsigned number.
- void **test_print** (const char *msgp)
Prints a line without final end-of-line.
- void **test_PRINTF** (const char *msgp)
Prints a line.

- void `test_emit_token` (char token)
Emits a token into the tokens buffer.
- void `test_terminate_threads` (void)
Sets a termination request in all the test-spawned threads.
- void `test_wait_threads` (void)
Waits for the completion of all the test-spawned threads.
- `systime_t test_wait_tick` (void)
Delays execution until next system time tick.
- void `test_start_timer` (unsigned ms)
Starts the test timer.

11.50.1 Detailed Description

Tests support header.

11.51 testbmk.c File Reference

Kernel Benchmarks source file.

```
#include "ch.h"
#include "test.h"
```

Variables

- ROMCONST struct testcase *ROMCONST patternbmk []
Test sequence for benchmarks.

11.51.1 Detailed Description

Kernel Benchmarks source file.

Kernel Benchmarks

11.51.2 Variable Documentation

11.51.2.1 ROMCONST struct testcase* ROMCONST patternbmk[]

Initial value:

```
= {  
  
    &testbmk1,  
    &testbmk2,  
    &testbmk3,  
  
    &testbmk4,  
    &testbmk5,  
    &testbmk6,  
  
    &testbmk7,  
    &testbmk8,  
    &testbmk9,  
    &testbmk10,
```

```
&testbmkl1,  
  
&testbmkl2,  
  
&testbmkl3,  
NULL  
}
```

Test sequence for benchmarks.

11.52 testbmk.h File Reference

Kernel Benchmarks header file.

Variables

- ROMCONST struct testcase *ROMCONST patternbmk []

Test sequence for benchmarks.

11.52.1 Detailed Description

Kernel Benchmarks header file.

11.52.2 Variable Documentation

11.52.2.1 ROMCONST struct testcase* ROMCONST patternbmk[]

Test sequence for benchmarks.

11.53 testdyn.c File Reference

Dynamic thread APIs test source file.

```
#include "ch.h"  
#include "test.h"
```

Variables

- ROMCONST struct testcase *ROMCONST patterndyn []

Test sequence for dynamic APIs.

11.53.1 Detailed Description

Dynamic thread APIs test source file.

11.53.2 Variable Documentation

11.53.2.1 ROMCONST struct testcase* ROMCONST patterndyn[]

Initial value:

```
= {  
    &testdyn1,  
  
    &testdyn2,  
  
    &testdyn3,  
  
    NULL  
}
```

Test sequence for dynamic APIs.

11.54 testdyn.h File Reference

Dynamic thread APIs test header file.

Variables

- ROMCONST struct testcase *ROMCONST patterndyn []
Test sequence for dynamic APIs.

11.54.1 Detailed Description

Dynamic thread APIs test header file.

11.54.2 Variable Documentation

11.54.2.1 ROMCONST struct testcase* ROMCONST patterndyn[]

Test sequence for dynamic APIs.

11.55 testevt.c File Reference

Events test source file.

```
#include "ch.h"  
#include "test.h"
```

Variables

- ROMCONST struct testcase *ROMCONST patternevt []
Test sequence for events.

11.55.1 Detailed Description

Events test source file.

11.55.2 Variable Documentation

11.55.2.1 ROMCONST struct testcase* ROMCONST patternevt[]

Initial value:

```
= {  
    &testevt1,  
    &testevt2,  
    &testevt3,  
  
    NULL  
}
```

Test sequence for events.

11.56 testevt.h File Reference

Events test header file.

Variables

- ROMCONST struct testcase *ROMCONST patternevt []

Test sequence for events.

11.56.1 Detailed Description

Events test header file.

11.56.2 Variable Documentation

11.56.2.1 ROMCONST struct testcase* ROMCONST patternevt[]

Test sequence for events.

11.57 testheap.c File Reference

Heap test source file.

```
#include "ch.h"  
#include "test.h"
```

Variables

- ROMCONST struct testcase *ROMCONST patternheap []

Test sequence for heap.

11.57.1 Detailed Description

Heap test source file.

11.57.2 Variable Documentation

11.57.2.1 ROMCONST struct testcase* ROMCONST patternheap[]

Initial value:

```
= {  
    &testheap1,  
    NULL  
}
```

Test sequence for heap.

11.58 testheap.h File Reference

Heap header file.

Variables

- ROMCONST struct testcase *ROMCONST patternheap []
Test sequence for heap.

11.58.1 Detailed Description

Heap header file.

11.58.2 Variable Documentation

11.58.2.1 ROMCONST struct testcase* ROMCONST patternheap[]

Test sequence for heap.

11.59 testmbox.c File Reference

Mailboxes test source file.

```
#include "ch.h"  
#include "test.h"
```

Variables

- ROMCONST struct testcase *ROMCONST patternmbox []
Test sequence for mailboxes.

11.59.1 Detailed Description

Mailboxes test source file.

11.59.2 Variable Documentation

11.59.2.1 ROMCONST struct testcase* ROMCONST patternmbox[]

Initial value:

```
= {  
    &testmbox1,  
    NULL  
}
```

Test sequence for mailboxes.

11.60 testmbox.h File Reference

Mailboxes header file.

Variables

- ROMCONST struct testcase *ROMCONST patternmbox []
Test sequence for mailboxes.

11.60.1 Detailed Description

Mailboxes header file.

11.60.2 Variable Documentation

11.60.2.1 ROMCONST struct testcase* ROMCONST patternmbox[]

Test sequence for mailboxes.

11.61 testmsg.c File Reference

Messages test source file.

```
#include "ch.h"  
#include "test.h"
```

Variables

- ROMCONST struct testcase *ROMCONST patternmsg []
Test sequence for messages.

11.61.1 Detailed Description

Messages test source file.

11.61.2 Variable Documentation

11.61.2.1 ROMCONST struct testcase* ROMCONST patternmsg[]

Initial value:

```
= {  
    &testmsg1,  
    NULL  
}
```

Test sequence for messages.

11.62 testmsg.h File Reference

Messages header file.

Variables

- ROMCONST struct testcase *ROMCONST patternmsg []
Test sequence for messages.

11.62.1 Detailed Description

Messages header file.

11.62.2 Variable Documentation

11.62.2.1 ROMCONST struct testcase* ROMCONST patternmsg[]

Test sequence for messages.

11.63 testmtx.c File Reference

Mutexes and CondVars test source file.

```
#include "ch.h"  
#include "test.h"
```

Variables

- ROMCONST struct testcase *ROMCONST patternmtx []
Test sequence for mutexes.

11.63.1 Detailed Description

Mutexes and CondVars test source file.

11.63.2 Variable Documentation

11.63.2.1 ROMCONST struct testcase* ROMCONST patternmtx[]

Initial value:

```
= {  
    &testmtx1,  
    &testmtx2,  
    &testmtx3,  
    &testmtx4,  
    &testmtx5,  
    &testmtx6,  
    &testmtx7,  
    &testmtx8,  
  
    NULL  
}
```

Test sequence for mutexes.

11.64 testmtx.h File Reference

Mutexes and CondVars test header file.

Variables

- ROMCONST struct testcase *ROMCONST patternmtx []
Test sequence for mutexes.

11.64.1 Detailed Description

Mutexes and CondVars test header file.

11.64.2 Variable Documentation

11.64.2.1 ROMCONST struct testcase* ROMCONST patternmtx[]

Test sequence for mutexes.

11.65 testpools.c File Reference

Memory Pools test source file.

```
#include "ch.h"  
#include "test.h"
```

11.65.1 Detailed Description

Memory Pools test source file.

11.66 testpools.h File Reference

Memory Pools test header file.

11.66.1 Detailed Description

Memory Pools test header file.

11.67 testqueues.c File Reference

I/O Queues test source file.

```
#include "ch.h"
#include "test.h"
```

Variables

- ROMCONST struct testcase *ROMCONST patternqueues []

Test sequence for queues.

11.67.1 Detailed Description

I/O Queues test source file.

11.67.2 Variable Documentation

11.67.2.1 ROMCONST struct testcase* ROMCONST patternqueues[]

Initial value:

```
= {
    &testqueues1,
    &testqueues2,
    NULL
}
```

Test sequence for queues.

11.68 testqueues.h File Reference

I/O Queues test header file.

Variables

- ROMCONST struct testcase *ROMCONST patternqueues []
Test sequence for queues.

11.68.1 Detailed Description

I/O Queues test header file.

11.68.2 Variable Documentation

11.68.2.1 ROMCONST struct testcase* ROMCONST patternqueues[]

Test sequence for queues.

11.69 testsem.c File Reference

Semaphores test source file.

```
#include "ch.h"
#include "test.h"
```

Variables

- ROMCONST struct testcase *ROMCONST patternsem []
Test sequence for semaphores.

11.69.1 Detailed Description

Semaphores test source file.

11.69.2 Variable Documentation

11.69.2.1 ROMCONST struct testcase* ROMCONST patternsem[]

Initial value:

```
= {
    &testsem1,
    &testsem2,
    &testsem3,
    &testsem4,
    NULL
}
```

Test sequence for semaphores.

11.70 testsem.h File Reference

Semaphores test header file.

Variables

- ROMCONST struct testcase *ROMCONST patternsem []
Test sequence for semaphores.

11.70.1 Detailed Description

Semaphores test header file.

11.70.2 Variable Documentation

11.70.2.1 ROMCONST struct testcase* ROMCONST patternsem[]

Test sequence for semaphores.

11.71 testsys.c File Reference

System test source file.

```
#include "ch.h"
#include "test.h"
```

Variables

- ROMCONST struct testcase *ROMCONST patternsys []
Test sequence for messages.

11.71.1 Detailed Description

System test source file.

11.71.2 Variable Documentation

11.71.2.1 ROMCONST struct testcase* ROMCONST patternsys[]

Initial value:

```
= {
  &testsyst1,
  &testsyst2,
  &testsyst3,
  NULL
}
```

Test sequence for messages.

11.72 testsys.h File Reference

System header file.

Variables

- ROMCONST struct testcase *ROMCONST patternsys []

Test sequence for messages.

11.72.1 Detailed Description

System header file.

11.72.2 Variable Documentation

11.72.2.1 ROMCONST struct testcase* ROMCONST patternsys[]

Test sequence for messages.

11.73 testthd.c File Reference

Threads and Scheduler test source file.

```
#include "ch.h"
#include "test.h"
```

Variables

- ROMCONST struct testcase *ROMCONST patternthd []

Test sequence for threads.

11.73.1 Detailed Description

Threads and Scheduler test source file.

11.73.2 Variable Documentation

11.73.2.1 ROMCONST struct testcase* ROMCONST patternthd[]

Initial value:

```
= {
  &testthd1,
  &testthd2,
  &testthd3,
  &testthd4,
  NULL
}
```

Test sequence for threads.

11.74 testthd.h File Reference

Threads and Scheduler test header file.

Variables

- ROMCONST struct testcase *ROMCONST patternthd []
Test sequence for threads.

11.74.1 Detailed Description

Threads and Scheduler test header file.

11.74.2 Variable Documentation

11.74.2.1 ROMCONST struct testcase* ROMCONST patternthd[]

Test sequence for threads.

Index

- _BSEMAPHORE_DATA
 - Binary Semaphores, [157](#)
- _CHIBIOS_RT_
 - Version Numbers and Identification, [38](#)
- _CONDVAR_DATA
 - Condition Variables, [179](#)
- _EVENTSOURCE_DATA
 - Event Flags, [190](#)
- _INPUTQUEUE_DATA
 - I/O Queues, [233](#)
- _MAILBOX_DATA
 - Mailboxes, [216](#)
- _MEMORYPOOL_DATA
 - Memory Pools, [263](#)
- _MUTEX_DATA
 - Mutexes, [168](#)
- _OUTPUTQUEUE_DATA
 - I/O Queues, [234](#)
- _SEMAPHORE_DATA
 - Counting Semaphores, [142](#)
- _THREADS_QUEUE_DATA
 - Threads, [97](#)
- _THREADS_QUEUE_DECL
 - Threads, [97](#)
- _base_sequential_stream_data
 - Abstract Sequential Streams, [275](#)
- _base_sequential_stream_methods
 - Abstract Sequential Streams, [275](#)
- _core_init
 - Core Memory Manager, [255](#)
- _dbg_check_disable
 - Debug, [286](#)
- _dbg_check_enable
 - Debug, [287](#)
- _dbg_check_enter_isr
 - Debug, [288](#)
- _dbg_check_leave_isr
 - Debug, [289](#)
- _dbg_check_lock
 - Debug, [287](#)
- _dbg_check_lock_from_isr
 - Debug, [288](#)
- _dbg_check_suspend
 - Debug, [286](#)
- _dbg_check_unlock
 - Debug, [287](#)
- _dbg_check_unlock_from_isr
 - Debug, [288](#)
- _dbg_trace
 - Debug, [290](#)
- _dbg_trace_init
 - Debug, [290](#)
- _heap_init
 - Heaps, [259](#)
- _idle_thread
 - System Management, [61](#)
- _port_init
 - Port Layer, [302](#)
- _port_switch
 - Port Layer, [302](#)
- _scheduler_init
 - Scheduler, [82](#)
- _stats_ctxswc
 - Statistics, [296](#)
- _stats_increase_irq
 - Statistics, [295](#)
- _stats_init
 - Statistics, [295](#)
- _stats_start_measure_crit_isr
 - Statistics, [297](#)
- _stats_start_measure_crit_thd
 - Statistics, [296](#)
- _stats_stop_measure_crit_isr
 - Statistics, [297](#)
- _stats_stop_measure_crit_thd
 - Statistics, [296](#)
- _thread_init
 - Threads, [98](#)
- _thread_memfill
 - Threads, [99](#)
- _tm_init
 - Time Measurement, [292](#)
- _vt_init
 - Time and Virtual Timers, [127](#)
- ABSPRIO
 - Scheduler, [77](#)
- ALL_EVENTS
 - Event Flags, [190](#)
- Abstract Sequential Streams, [275](#)
 - _base_sequential_stream_data, [275](#)
 - _base_sequential_stream_methods, [275](#)
 - chSequentialStreamGet, [277](#)
 - chSequentialStreamPut, [276](#)
 - chSequentialStreamRead, [276](#)
 - chSequentialStreamWrite, [276](#)
- BSEMAPHORE_DECL
 - Binary Semaphores, [157](#)

Base Kernel Services, 53
 BaseSequentialStream, 315
 vmt, 316
 BaseSequentialStreamVMT, 316
 best
 time_measurement_t, 361
 Binary Semaphores, 156
 _BSEMAPHORE_DATA, 157
 BSEMAPHORE_DECL, 157
 chBSemGetStatel, 165
 chBSemObjectInit, 157
 chBSemReset, 162
 chBSemResetl, 162
 chBSemSignal, 164
 chBSemSignall, 163
 chBSemWait, 158
 chBSemWaitS, 158
 chBSemWaitTimeout, 161
 chBSemWaitTimeoutS, 160
 binary_semaphore_t, 316
 c_queue
 condition_variable, 345
 CH_CFG_CONTEXT_SWITCH_HOOK
 Configuration, 48
 CH_CFG_IDLE_ENTER_HOOK
 Configuration, 48
 CH_CFG_IDLE_LEAVE_HOOK
 Configuration, 48
 CH_CFG_IDLE_LOOP_HOOK
 Configuration, 49
 CH_CFG_MEMCORE_SIZE
 Configuration, 41
 CH_CFG_NO_IDLE_THREAD
 Configuration, 42
 CH_CFG_OPTIMIZE_SPEED
 Configuration, 42
 CH_CFG_ST_FREQUENCY
 Configuration, 41
 CH_CFG_ST_RESOLUTION
 Configuration, 41
 CH_CFG_ST_TIMEDELTA
 Configuration, 41
 CH_CFG_SYSTEM_HALT_HOOK
 Configuration, 49
 CH_CFG_SYSTEM_TICK_HOOK
 Configuration, 49
 CH_CFG_THREAD_EXIT_HOOK
 Configuration, 48
 CH_CFG_THREAD_EXTRA_FIELDS
 Configuration, 47
 CH_CFG_THREAD_INIT_HOOK
 Configuration, 47
 CH_CFG_TIME_QUANTUM
 Configuration, 41
 CH_CFG_USE_CONDVARs
 Configuration, 43
 CH_CFG_USE_CONDVARs_TIMEOUT
 Configuration, 43
 CH_CFG_USE_DYNAMIC
 Configuration, 45
 CH_CFG_USE_EVENTS
 Configuration, 44
 CH_CFG_USE_EVENTS_TIMEOUT
 Configuration, 44
 CH_CFG_USE_HEAP
 Configuration, 45
 CH_CFG_USE_MAILBOXES
 Configuration, 44
 CH_CFG_USE_MEMCORE
 Configuration, 45
 CH_CFG_USE_MEMPOOLS
 Configuration, 45
 CH_CFG_USE_MESSAGES
 Configuration, 44
 CH_CFG_USE_MESSAGES_PRIORITY
 Configuration, 44
 CH_CFG_USE_MUTEXES
 Configuration, 43
 CH_CFG_USE_MUTEXES_RECURSIVE
 Configuration, 43
 CH_CFG_USE_QUEUES
 Configuration, 45
 CH_CFG_USE_REGISTRY
 Configuration, 42
 CH_CFG_USE_SEMAPHORES
 Configuration, 43
 CH_CFG_USE_SEMAPHORES_PRIORITY
 Configuration, 43
 CH_CFG_USE_TM
 Configuration, 42
 CH_CFG_USE_WAITEXIT
 Configuration, 42
 CH_CUSTOMER_ID_CODE
 Customer, 312
 CH_CUSTOMER_ID_STRING
 Customer, 312
 CH_DBG_ENABLE_ASSERTS
 Configuration, 46
 CH_DBG_ENABLE_CHECKS
 Configuration, 46
 CH_DBG_ENABLE_STACK_CHECK
 Configuration, 47
 CH_DBG_ENABLE_TRACE
 Configuration, 46
 CH_DBG_FILL_THREADS
 Configuration, 47
 CH_DBG_STACK_FILL_VALUE
 Debug, 285
 CH_DBG_STATISTICS
 Configuration, 46
 CH_DBG_SYSTEM_STATE_CHECK
 Configuration, 46
 CH_DBG_THREAD_FILL_VALUE
 Debug, 285
 CH_DBG_THREADS_PROFILING
 Configuration, 47

CH_DBG_TRACE_BUFFER_SIZE
 Debug, 285

CH_FAST_IRQ_HANDLER
 System Management, 57

CH_FLAG_MODE_HEAP
 Scheduler, 79

CH_FLAG_MODE_MASK
 Scheduler, 79

CH_FLAG_MODE_MPOOL
 Scheduler, 79

CH_FLAG_MODE_STATIC
 Scheduler, 79

CH_FLAG_TERMINATE
 Scheduler, 79

CH_IRQ_EPILOGUE
 System Management, 57

CH_IRQ_HANDLER
 System Management, 57

CH_IRQ_IS_VALID_KERNEL_PRIORITY
 System Management, 56

CH_IRQ_IS_VALID_PRIORITY
 System Management, 56

CH_IRQ_PROLOGUE
 System Management, 56

CH_KERNEL_MAJOR
 Version Numbers and Identification, 38

CH_KERNEL_MINOR
 Version Numbers and Identification, 38

CH_KERNEL_PATCH
 Version Numbers and Identification, 38

CH_KERNEL_STABLE
 Version Numbers and Identification, 38

CH_KERNEL_VERSION
 Version Numbers and Identification, 38

CH_LICENSE
 License, 313

CH_LICENSE_FEATURES
 License, 314

CH_LICENSE_ID_CODE
 License, 314

CH_LICENSE_ID_STRING
 License, 314

CH_LICENSE_MAX_DEPLOY
 License, 314

CH_LICENSE_MODIFIABLE_CODE
 License, 314

CH_LICENSE_TYPE_STRING
 License, 313

CH_STATE_CURRENT
 Scheduler, 78

CH_STATE_FINAL
 Scheduler, 79

CH_STATE_NAMES
 Scheduler, 79

CH_STATE_QUEUED
 Scheduler, 78

CH_STATE_READY
 Scheduler, 77

CH_STATE_SLEEPING
 Scheduler, 78

CH_STATE SNDMSG
 Scheduler, 79

CH_STATE SNDMSGQ
 Scheduler, 78

CH_STATE_SUSPENDED
 Scheduler, 78

CH_STATE_WTANDEVT
 Scheduler, 78

CH_STATE_WTCOND
 Scheduler, 78

CH_STATE_WTEXIT
 Scheduler, 78

CH_STATE_WTMSG
 Scheduler, 79

CH_STATE_WTMTX
 Scheduler, 78

CH_STATE_WTOREVT
 Scheduler, 78

CH_STATE_WTSEM
 Scheduler, 78

CH_STATE_WTSTART
 Scheduler, 78

CONDVAR_DECL
 Condition Variables, 179

cf_off_ctx
 chdebug_t, 343

cf_off_flags
 chdebug_t, 343

cf_off_name
 chdebug_t, 343

cf_off_newer
 chdebug_t, 343

cf_off_older
 chdebug_t, 343

cf_off_preempt
 chdebug_t, 344

cf_off_prio
 chdebug_t, 343

cf_off_refs
 chdebug_t, 343

cf_off_state
 chdebug_t, 343

cf_off_stklimit
 chdebug_t, 343

cf_off_time
 chdebug_t, 344

ch
 Scheduler, 94

ch.h, 363

ch_identifier
 chdebug_t, 342

ch_mutex, 318

 m_cnt, 320

 m_next, 320

 m_owner, 320

 m_queue, 320

ch_ptrsize
 chdebug_t, 343
 ch_semaphore, 320
 s_cnt, 321
 s_queue, 321
 ch_size
 chdebug_t, 342
 ch_swc_event_t, 322
 se_state, 323
 se_time, 323
 se_tp, 323
 se_wtobjp, 323
 ch_system, 323
 dbg, 324
 kernel_stats, 325
 mainthread, 324
 rlist, 324
 THD_WORKING_AREA, 324
 tm, 324
 vtlist, 324
 ch_system_debug, 325
 isr_cnt, 326
 lock_cnt, 326
 panic_msg, 326
 trace_buffer, 326
 ch_system_t
 Scheduler, 82
 ch_thread, 326
 ewmask, 331
 exitcode, 330
 p_ctx, 329
 p_epending, 331
 p_flags, 329
 p_mpool, 332
 p_msg, 331
 p_msgqueue, 331
 p_mtxlist, 331
 p_name, 329
 p_newer, 329
 p_next, 329
 p_older, 329
 p_preempt, 330
 p_prev, 329
 p_prio, 329
 p_realprio, 332
 p_refs, 329
 p_state, 329
 p_stats, 332
 p_stklimit, 329
 p_time, 330
 p_u, 331
 p_waiting, 331
 rdymsg, 330
 wtmtxp, 331
 wtobjp, 330
 wtsemp, 330
 wttrp, 330
 ch_threads_list, 332
 p_next, 334
 ch_threads_queue, 334
 p_next, 335
 p_prev, 335
 ch_threadsize
 chdebug_t, 343
 ch_timesize
 chdebug_t, 343
 ch_trace_buffer_t, 335
 tb_buffer, 337
 tb_ptr, 337
 tb_size, 337
 ch_version
 chdebug_t, 342
 ch_virtual_timer, 337
 vt_delta, 338
 vt_func, 339
 vt_next, 338
 vt_par, 339
 vt_prev, 338
 ch_virtual_timers_list, 339
 vt_delta, 341
 vt_lasttime, 341
 vt_next, 340
 vt_prev, 340
 vt_systime, 341
 ch_zero
 chdebug_t, 342
 chBSemGetStateI
 Binary Semaphores, 165
 chBSemObjectInit
 Binary Semaphores, 157
 chBSemReset
 Binary Semaphores, 162
 chBSemResetI
 Binary Semaphores, 162
 chBSemSignal
 Binary Semaphores, 164
 chBSemSignall
 Binary Semaphores, 163
 chBSemWait
 Binary Semaphores, 158
 chBSemWaitS
 Binary Semaphores, 158
 chBSemWaitTimeout
 Binary Semaphores, 161
 chBSemWaitTimeoutS
 Binary Semaphores, 160
 chCondBroadcast
 Condition Variables, 181
 chCondBroadcastI
 Condition Variables, 182
 chCondObjectInit
 Condition Variables, 179
 chCondSignal
 Condition Variables, 179
 chCondSignall
 Condition Variables, 180

chCondWait
 Condition Variables, 183
chCondWaitS
 Condition Variables, 184
chCondWaitTimeout
 Condition Variables, 185
chCondWaitTimeoutS
 Condition Variables, 186
chCoreAlloc
 Core Memory Manager, 255
chCoreAllocI
 Core Memory Manager, 256
chCoreGetStatusX
 Core Memory Manager, 257
chDbgAssert
 Debug, 285
chDbgCheck
 Debug, 285
chDbgCheckClassI
 Debug, 289
chDbgCheckClassS
 Debug, 290
chEvtAddEvents
 Event Flags, 193
chEvtBroadcast
 Event Flags, 207
chEvtBroadcastFlags
 Event Flags, 197
chEvtBroadcastFlagsI
 Event Flags, 194
chEvtBroadcastI
 Event Flags, 207
chEvtDispatch
 Event Flags, 198
chEvtGetAndClearEvents
 Event Flags, 192
chEvtGetAndClearFlags
 Event Flags, 195
chEvtGetAndClearFlagsI
 Event Flags, 198
chEvtIsListeningI
 Event Flags, 207
chEvtObjectInit
 Event Flags, 205
chEvtRegister
 Event Flags, 206
chEvtRegisterMask
 Event Flags, 205
chEvtRegisterMaskWithFlags
 Event Flags, 190
chEvtSignal
 Event Flags, 195
chEvtSignall
 Event Flags, 196
chEvtUnregister
 Event Flags, 191
chEvtWaitAll
 Event Flags, 201
chEvtWaitAllTimeout
 Event Flags, 204
chEvtWaitAny
 Event Flags, 200
chEvtWaitAnyTimeout
 Event Flags, 203
chEvtWaitOne
 Event Flags, 199
chEvtWaitOneTimeout
 Event Flags, 202
chHeapAlloc
 Heaps, 260
chHeapFree
 Heaps, 260
chHeapObjectInit
 Heaps, 259
chHeapStatus
 Heaps, 261
chIQGet
 I/O Queues, 249
chIQGetEmptyI
 I/O Queues, 247
chIQGetFullI
 I/O Queues, 246
chIQGetTimeout
 I/O Queues, 239
chIQIsEmptyI
 I/O Queues, 247
chIQIsFullI
 I/O Queues, 248
chIQObjectInit
 I/O Queues, 236
chIQPutI
 I/O Queues, 237
chIQReadTimeout
 I/O Queues, 240
chIQResetI
 I/O Queues, 237
chMBFetch
 Mailboxes, 225
chMBFetchI
 Mailboxes, 227
chMBFetchS
 Mailboxes, 226
chMBGetFreeCountI
 Mailboxes, 228
chMBGetSizeI
 Mailboxes, 228
chMBGetUsedCountI
 Mailboxes, 229
chMBOObjectInit
 Mailboxes, 217
chMBPeekI
 Mailboxes, 229
chMBPost
 Mailboxes, 219
chMBPostAhead
 Mailboxes, 222

chMBPostAheadI
 Mailboxes, 224
 chMBPostAheadS
 Mailboxes, 223
 chMBPostI
 Mailboxes, 221
 chMBPostS
 Mailboxes, 220
 chMBReset
 Mailboxes, 217
 chMBResetI
 Mailboxes, 218
 chMsgGet
 Synchronous Messages, 213
 chMsgIsPendingI
 Synchronous Messages, 212
 chMsgRelease
 Synchronous Messages, 211
 chMsgReleaseS
 Synchronous Messages, 213
 chMsgSend
 Synchronous Messages, 209
 chMsgWait
 Synchronous Messages, 210
 chMtxGetNextMutexS
 Mutexes, 177
 chMtxLock
 Mutexes, 169
 chMtxLockS
 Mutexes, 170
 chMtxObjectInit
 Mutexes, 169
 chMtxQueueNotEmptyS
 Mutexes, 176
 chMtxTryLock
 Mutexes, 171
 chMtxTryLockS
 Mutexes, 172
 chMtxUnlock
 Mutexes, 173
 chMtxUnlockAll
 Mutexes, 175
 chMtxUnlockS
 Mutexes, 174
 chOQGetEmptyI
 I/O Queues, 250
 chOQGetFullI
 I/O Queues, 249
 chOQGetI
 I/O Queues, 244
 chOQIsEmptyI
 I/O Queues, 251
 chOQIsFullI
 I/O Queues, 251
 chOQObjectInit
 I/O Queues, 241
 chOQPut
 I/O Queues, 252
 chOQPutTimeout
 I/O Queues, 243
 chOQResetI
 I/O Queues, 242
 chOQWriteTimeout
 I/O Queues, 245
 chPoolAdd
 Memory Pools, 267
 chPoolAddI
 Memory Pools, 268
 chPoolAlloc
 Memory Pools, 264
 chPoolAllocI
 Memory Pools, 264
 chPoolFree
 Memory Pools, 266
 chPoolFreeI
 Memory Pools, 265
 chPoolLoadArray
 Memory Pools, 263
 chPoolObjectInit
 Memory Pools, 263
 chQGetLinkX
 I/O Queues, 235
 chQSizeX
 I/O Queues, 235
 chQSpaceI
 I/O Queues, 235
 chRegFirstThread
 Registry, 279
 chRegGetThreadNameX
 Registry, 281
 chRegNextThread
 Registry, 280
 chRegSetThreadName
 Registry, 281
 chRegSetThreadNameX
 Registry, 282
 chSchCanYieldS
 Scheduler, 93
 chSchDoReschedule
 Scheduler, 89
 chSchDoRescheduleAhead
 Scheduler, 89
 chSchDoRescheduleBehind
 Scheduler, 88
 chSchDoYieldS
 Scheduler, 93
 chSchGoSleepS
 Scheduler, 85
 chSchGoSleepTimeoutS
 Scheduler, 86
 chSchIsPreemptionRequired
 Scheduler, 88
 chSchIsRescRequiredI
 Scheduler, 92
 chSchPreemption
 Scheduler, 94

chSchReadyI
 Scheduler, 84
chSchRescheduleS
 Scheduler, 87
chSchWakeupS
 Scheduler, 87
chSemAddCounterI
 Counting Semaphores, 151
chSemFastSignall
 Counting Semaphores, 154
chSemFastWaitI
 Counting Semaphores, 153
chSemGetCounterI
 Counting Semaphores, 154
chSemObjectInit
 Counting Semaphores, 143
chSemReset
 Counting Semaphores, 143
chSemResetI
 Counting Semaphores, 144
chSemSignal
 Counting Semaphores, 149
chSemSignall
 Counting Semaphores, 150
chSemSignalWait
 Counting Semaphores, 151
chSemWait
 Counting Semaphores, 145
chSemWaitS
 Counting Semaphores, 146
chSemWaitTimeout
 Counting Semaphores, 147
chSemWaitTimeoutS
 Counting Semaphores, 148
chSequentialStreamGet
 Abstract Sequential Streams, 277
chSequentialStreamPut
 Abstract Sequential Streams, 276
chSequentialStreamRead
 Abstract Sequential Streams, 276
chSequentialStreamWrite
 Abstract Sequential Streams, 276
chSysDisable
 System Management, 67
chSysEnable
 System Management, 68
chSysGetIdleThreadX
 System Management, 72
chSysGetRealtimeCounterX
 System Management, 60
chSysGetStatusAndLockX
 System Management, 64
chSysHalt
 System Management, 62
chSysInit
 System Management, 61
chSysIntegrityCheckI
 System Management, 63
chSysIsCounterWithinX
 System Management, 66
chSysLock
 System Management, 69
chSysLockFromISR
 System Management, 70
chSysPolledDelayX
 System Management, 67
chSysRestoreStatusX
 System Management, 65
chSysSuspend
 System Management, 68
chSysSwitch
 System Management, 60
chSysTimerHandlerI
 System Management, 64
chSysUnconditionalLock
 System Management, 71
chSysUnconditionalUnlock
 System Management, 71
chSysUnlock
 System Management, 69
chSysUnlockFromISR
 System Management, 70
chTMChainMeasurementToX
 Time Measurement, 294
chTMOBJECTINIT
 Time Measurement, 292
chTMStartMeasurementX
 Time Measurement, 294
chTMStopMeasurementX
 Time Measurement, 294
chThdAddRef
 Dynamic Threads, 269
chThdCreateFromHeap
 Dynamic Threads, 271
chThdCreateFromMemoryPool
 Dynamic Threads, 272
chThdCreateI
 Threads, 99
chThdCreateStatic
 Threads, 100
chThdDequeueAllI
 Threads, 116
chThdDequeueNextI
 Threads, 116
chThdDoDequeueNextI
 Threads, 121
chThdEnqueueTimeoutS
 Threads, 115
chThdExit
 Threads, 108
chThdExitS
 Threads, 109
chThdGetPriorityX
 Threads, 117
chThdGetSelfX
 Threads, 117

chThdGetTicksX
 Threads, 118
chThdQueueIsEmpty
 Threads, 120
chThdQueueObjectInit
 Threads, 120
chThdRelease
 Dynamic Threads, 270
chThdResume
 Threads, 114
chThdResumel
 Threads, 113
chThdResumeS
 Threads, 113
chThdSetPriority
 Threads, 102
chThdShouldTerminateX
 Threads, 118
chThdSleep
 Threads, 104
chThdSleepMicroseconds
 Threads, 98
chThdSleepMilliseconds
 Threads, 97
chThdSleepSeconds
 Threads, 97
chThdSleepUntil
 Threads, 105
chThdSleepUntilWindowed
 Threads, 106
chThdStart
 Threads, 101
chThdStartl
 Threads, 119
chThdSuspendS
 Threads, 111
chThdSuspendTimeoutS
 Threads, 112
chThdTerminate
 Threads, 103
chThdTerminatedX
 Threads, 118
chThdWait
 Threads, 110
chThdYield
 Threads, 107
chVTDoResetl
 Time and Virtual Timers, 128
chVTDoSetl
 Time and Virtual Timers, 127
chVTDoTickl
 Time and Virtual Timers, 138
chVTGetSystemTime
 Time and Virtual Timers, 129
chVTGetSystemTimeX
 Time and Virtual Timers, 129
chVTGetTimersStateI
 Time and Virtual Timers, 132
chVtIsArmed
 Time and Virtual Timers, 133
chVtIsArmedl
 Time and Virtual Timers, 133
chVtIsSystemTimeWithin
 Time and Virtual Timers, 131
chVtIsSystemTimeWithinX
 Time and Virtual Timers, 131
chVtIsTimeWithinX
 Time and Virtual Timers, 130
chVtObjectInit
 Time and Virtual Timers, 128
chVtReset
 Time and Virtual Timers, 135
chVtResetl
 Time and Virtual Timers, 134
chVtSet
 Time and Virtual Timers, 137
chVtSetl
 Time and Virtual Timers, 136
chVtTimeElapsedSinceX
 Time and Virtual Timers, 130
chbsem.h, 364
chcond.c, 365
chcond.h, 365
chconf.h, 366
chcore.c, 368
chcore.h, 369
chcustomer.h, 371
chdebug.c, 371
chdebug.h, 372
chdebug_t, 341

- cf_off_ctxt, 343
- cf_off_flags, 343
- cf_off_name, 343
- cf_off_newer, 343
- cf_off_older, 343
- cf_off_preempt, 344
- cf_off_prio, 343
- cf_off_refs, 343
- cf_off_state, 343
- cf_off_stklimit, 343
- cf_off_time, 344
- ch_identifier, 342
- ch_ptrsize, 343
- ch_size, 342
- ch_threadsize, 343
- ch_timesize, 343
- ch_version, 342
- ch_zero, 342

chdynamic.c, 372
chdynamic.h, 373
chevents.c, 373
chevents.h, 374
chheap.c, 376
chheap.h, 377

chlicense.h, 377
chmboxes.c, 378
chmboxes.h, 379
chmemcore.c, 380
chmemcore.h, 380
chmempools.c, 381
chmempools.h, 382
chmsg.c, 383
chmsg.h, 383
chmtx.c, 384
chmtx.h, 384
chqueues.c, 385
chqueues.h, 386
chregistry.c, 388
chregistry.h, 388
chsched.c, 389
chsched.h, 390
chsem.c, 393
chsem.h, 394
chstats.c, 395
chstats.h, 396
chstreams.h, 396
chsys.c, 397
chsys.h, 397
chsysypes.h, 399
chthreads.c, 400
chthreads.h, 401
chtm.c, 403
chtm.h, 403
ctypes.h, 404
chvt.c, 405
chvt.h, 405
cnt_t
 Kernel Types, 52
Condition Variables, 178
 _CONDVAR_DATA, 179
 CONDVAR_DECL, 179
 chCondBroadcast, 181
 chCondBroadcastI, 182
 chCondObjectInit, 179
 chCondSignal, 179
 chCondSignall, 180
 chCondWait, 183
 chCondWaitS, 184
 chCondWaitTimeout, 185
 chCondWaitTimeoutS, 186
 condition_variable_t, 179
condition_variable, 344
 c_queue, 345
condition_variable_t
 Condition Variables, 179
Configuration, 39
 CH_CFG_CONTEXT_SWITCH_HOOK, 48
 CH_CFG_IDLE_ENTER_HOOK, 48
 CH_CFG_IDLE_LEAVE_HOOK, 48
 CH_CFG_IDLE_LOOP_HOOK, 49
 CH_CFG_MEMCORE_SIZE, 41
 CH_CFG_NO_IDLE_THREAD, 42
 CH_CFG_OPTIMIZE_SPEED, 42
 CH_CFG_ST_FREQUENCY, 41
 CH_CFG_ST_RESOLUTION, 41
 CH_CFG_ST_TIMEDELTA, 41
 CH_CFG_SYSTEM_HALT_HOOK, 49
 CH_CFG_SYSTEM_TICK_HOOK, 49
 CH_CFG_THREAD_EXIT_HOOK, 48
 CH_CFG_THREAD_EXTRA_FIELDS, 47
 CH_CFG_THREAD_INIT_HOOK, 47
 CH_CFG_TIME_QUANTUM, 41
 CH_CFG_USE_CONDVARS, 43
 CH_CFG_USE_CONDVARS_TIMEOUT, 43
 CH_CFG_USE_DYNAMIC, 45
 CH_CFG_USE_EVENTS, 44
 CH_CFG_USE_EVENTS_TIMEOUT, 44
 CH_CFG_USE_HEAP, 45
 CH_CFG_USE_MAILBOXES, 44
 CH_CFG_USE_MEMCORE, 45
 CH_CFG_USE_MEMPOOLS, 45
 CH_CFG_USE_MESSAGES, 44
 CH_CFG_USE_MESSAGES_PRIORITY, 44
 CH_CFG_USE_MUTEXES, 43
 CH_CFG_USE_MUTEXES_RECURSIVE, 43
 CH_CFG_USE_QUEUES, 45
 CH_CFG_USE_REGISTRY, 42
 CH_CFG_USE_SEMAPHORES, 43
 CH_CFG_USE_SEMAPHORES_PRIORITY, 43
 CH_CFG_USE_TM, 42
 CH_CFG_USE_WAITEXIT, 42
 CH_DBG_ENABLE_ASSERTS, 46
 CH_DBG_ENABLE_CHECKS, 46
 CH_DBG_ENABLE_STACK_CHECK, 47
 CH_DBG_ENABLE_TRACE, 46
 CH_DBG_FILL_THREADS, 47
 CH_DBG_STATISTICS, 46
 CH_DBG_SYSTEM_STATE_CHECK, 46
 CH_DBG_THREADS_PROFILING, 47
context, 345
Core Memory Manager, 254
 _core_init, 255
 chCoreAlloc, 255
 chCoreAllocI, 256
 chCoreGetStatusX, 257
 MEM_ALIGN_MASK, 255
 MEM_ALIGN_NEXT, 255
 MEM_ALIGN_PREV, 255
 MEM_ALIGN_SIZE, 255
 MEM_IS_ALIGNED, 255
 memgetfunc_t, 255
Counting Semaphores, 141
 _SEMAPHORE_DATA, 142
 chSemAddCounterI, 151
 chSemFastSignall, 154
 chSemFastWaitI, 153
 chSemGetCounterI, 154
 chSemObjectInit, 143
 chSemReset, 143
 chSemResetI, 144

chSemSignal, 149
chSemSignall, 150
chSemSignalWait, 151
chSemWait, 145
chSemWaitS, 146
chSemWaitTimeout, 147
chSemWaitTimeoutS, 148
SEMAPHORE_DECL, 142
semaphore_t, 143
cumulative
 time_measurement_t, 361
currp
 Scheduler, 80
Customer, 312
 CH_CUSTOMER_ID_CODE, 312
 CH_CUSTOMER_ID_STRING, 312
DELAY_BETWEEN_TESTS
 Test Runtime, 306
dbg
 ch_system, 324
Debug, 283
 _dbg_check_disable, 286
 _dbg_check_enable, 287
 _dbg_check_enter_isr, 288
 _dbg_check_leave_isr, 289
 _dbg_check_lock, 287
 _dbg_check_lock_from_isr, 288
 _dbg_check_suspend, 286
 _dbg_check_unlock, 287
 _dbg_check_unlock_from_isr, 288
 _dbg_trace, 290
 _dbg_trace_init, 290
 CH_DBG_STACK_FILL_VALUE, 285
 CH_DBG_THREAD_FILL_VALUE, 285
 CH_DBG_TRACE_BUFFER_SIZE, 285
 chDbgAssert, 285
 chDbgCheck, 285
 chDbgCheckClassI, 289
 chDbgCheckClassS, 290
default_heap
 Heaps, 261
Dynamic Threads, 269
 chThdAddRef, 269
 chThdCreateFromHeap, 271
 chThdCreateFromMemoryPool, 272
 chThdRelease, 270
EVENT_MASK
 Event Flags, 190
EVENTSOURCE_DECL
 Event Flags, 190
el_events
 event_listener, 347
el_flags
 event_listener, 347
el_listener
 event_listener, 346
el_next

event_listener, 346
el_wflags
 event_listener, 347
es_next
 event_source, 348
Event Flags, 188
 _EVENTSOURCE_DATA, 190
 ALL_EVENTS, 190
 chEvtAddEvents, 193
 chEvtBroadcast, 207
 chEvtBroadcastFlags, 197
 chEvtBroadcastFlagsI, 194
 chEvtBroadcastI, 207
 chEvtDispatch, 198
 chEvtGetAndClearEvents, 192
 chEvtGetAndClearFlags, 195
 chEvtGetAndClearFlagsI, 198
 chEvtIsListeningI, 207
 chEvtObjectInit, 205
 chEvtRegister, 206
 chEvtRegisterMask, 205
 chEvtRegisterMaskWithFlags, 190
 chEvtSignal, 195
 chEvtSignall, 196
 chEvtUnregister, 191
 chEvtWaitAll, 201
 chEvtWaitAllTimeout, 204
 chEvtWaitAny, 200
 chEvtWaitAnyTimeout, 203
 chEvtWaitOne, 199
 chEvtWaitOneTimeout, 202
 EVENT_MASK, 190
 EVENTSOURCE_DECL, 190
 event_source_t, 190
 evhandler_t, 190
event_listener, 345
 el_events, 347
 el_flags, 347
 el_listener, 346
 el_next, 346
 el_wflags, 347
event_source, 347
 es_next, 348
event_source_t
 Event Flags, 190
eventflags_t
 Kernel Types, 52
eventid_t
 Kernel Types, 52
eventmask_t
 Kernel Types, 52
evhandler_t
 Event Flags, 190
ewmask
 ch_thread, 331
execute
 testcase, 360
exitcode

ch_thread, 330
FALSE
 Kernel Types, 50
firstprio
 Scheduler, 80

h_free
 memory_heap, 356
h_mtx
 memory_heap, 356
h_provider
 memory_heap, 356
HIGHPRIO
 Scheduler, 77
heap
 heap_header, 349
heap_header, 349
 heap, 349
 next, 349
 size, 349
 u, 349
Heaps, 258
 _heap_init, 259
 chHeapAlloc, 260
 chHeapFree, 260
 chHeapObjectInit, 259
 chHeapStatus, 261
 default_heap, 261
 memory_heap_t, 258

I/O Queues, 231
 _INPUTQUEUE_DATA, 233
 _OUTPUTQUEUE_DATA, 234
 chIQGet, 249
 chIQGetEmptyl, 247
 chIQGetFulll, 246
 chIQGetTimeout, 239
 chIQIsEmptyl, 247
 chIQIsFulll, 248
 chIQObjectInit, 236
 chIQPutl, 237
 chIQReadTimeout, 240
 chIQResetl, 237
 chOQGetEmptyl, 250
 chOQGetFulll, 249
 chOQGetl, 244
 chOQIsEmptyl, 251
 chOQIsFulll, 251
 chOQObjectInit, 241
 chOQPut, 252
 chOQPutTimeout, 243
 chOQResetl, 242
 chOQWriteTimeout, 245
 chQGetLinkX, 235
 chQSizeX, 235
 chQSpacel, 235
INPUTQUEUE DECL, 234
input_queue_t, 236
 io_queue_t, 236
 OUTPUTQUEUE DECL, 234
 output_queue_t, 236
 Q_EMPTY, 233
 Q_FULL, 233
 Q_OK, 233
 Q_RESET, 233
 Q_TIMEOUT, 233
 qnotify_t, 236
IDLEPRIO
 Scheduler, 77
INPUTQUEUE DECL
 I/O Queues, 234
input_queue_t
 I/O Queues, 236
 io_queue, 350
 q_buffer, 351
 q_counter, 351
 q_link, 351
 q_notify, 351
 q_rptr, 351
 q_top, 351
 q_waiting, 351
 q_wptr, 351
 io_queue_t
 I/O Queues, 236
isr_cnt
 ch_system_debug, 326

Kernel Types, 50
 cnt_t, 52
 eventflags_t, 52
 eventid_t, 52
 eventmask_t, 52
 FALSE, 50
 msg_t, 52
 NOINLINE, 51
 PACKED_VAR, 51
 PORT THD FUNCTION, 51
 ROMCONST, 50
 rtcnt_t, 51
 rttime_t, 51
 syssts_t, 51
 TRUE, 50
 tmode_t, 51
 tprio_t, 52
 trefs_t, 51
 tslices_t, 51
 tstate_t, 51
 ucnt_t, 52
kernel_stats
 ch_system, 325
kernel_stats_t, 351
 m_crit_isr, 353
 m_crit_thd, 352
 n_ctxswc, 352
 n_irq, 352
LOWPRIO

Scheduler, 77
 last
 time_measurement_t, 361
 License, 313
 CH_LICENSE, 313
 CH_LICENSE_FEATURES, 314
 CH_LICENSE_ID_CODE, 314
 CH_LICENSE_ID_STRING, 314
 CH_LICENSE_MAX_DEPLOY, 314
 CH_LICENSE_MODIFIABLE_CODE, 314
 CH_LICENSE_TYPE_STRING, 313
 list_init
 Scheduler, 90
 list_insert
 Scheduler, 84
 list_isempty
 Scheduler, 90
 list_notempty
 Scheduler, 90
 list_remove
 Scheduler, 84
 lock_cnt
 ch_system_debug, 326

 m_cnt
 ch_mutex, 320
 m_crit_isr
 kernel_stats_t, 353
 m_crit_thd
 kernel_stats_t, 352
 m_next
 ch_mutex, 320
 m_owner
 ch_mutex, 320
 m_queue
 ch_mutex, 320
 MAILBOX_DECL
 Mailboxes, 216
 MEM_ALIGN_MASK
 Core Memory Manager, 255
 MEM_ALIGN_NEXT
 Core Memory Manager, 255
 MEM_ALIGN_PREV
 Core Memory Manager, 255
 MEM_ALIGN_SIZE
 Core Memory Manager, 255
 MEM_IS_ALIGNED
 Core Memory Manager, 255
 MEMORYPOOL_DECL
 Memory Pools, 263
 MS2RTC
 System Management, 58
 MS2ST
 Time and Virtual Timers, 124
 MSG_OK
 Scheduler, 77
 MSG_RESET
 Scheduler, 77
 MSG_TIMEOUT

Scheduler, 77
 MUTEX_DECL
 Mutexes, 168
 mailbox_t, 353
 mb_buffer, 354
 mb_emptysem, 354
 mb_fullsem, 354
 mb_rdptr, 354
 mb_top, 354
 mb_wrptr, 354
 Mailboxes, 215
 _MAILBOX_DATA, 216
 chMBFetch, 225
 chMBFetchl, 227
 chMBFetchS, 226
 chMBGetFreeCountl, 228
 chMBGetSizeL, 228
 chMBGetUsedCountl, 229
 chMBOBJECTInit, 217
 chMBPeekl, 229
 chMBPost, 219
 chMBPostAhead, 222
 chMBPostAheadl, 224
 chMBPostAheadS, 223
 chMBPostl, 221
 chMBPostS, 220
 chMBReset, 217
 chMBResetl, 218
 MAILBOX_DECL, 216
 mainthread
 ch_system, 324
 mb_buffer
 mailbox_t, 354
 mb_emptysem
 mailbox_t, 354
 mb_fullsem
 mailbox_t, 354
 mb_rdptr
 mailbox_t, 354
 mb_top
 mailbox_t, 354
 mb_wrptr
 mailbox_t, 354
 memgetfunc_t
 Core Memory Manager, 255
 Memory Management, 253
 Memory Pools, 262
 _MEMORYPOOL_DATA, 263
 chPoolAdd, 267
 chPoolAddl, 268
 chPoolAlloc, 264
 chPoolAllocl, 264
 chPoolFree, 266
 chPoolFreel, 265
 chPoolLoadArray, 263
 chPoolObjectInit, 263
 MEMORYPOOL_DECL, 263
 memory_heap, 354

h_free, 356
h_mtx, 356
h_provider, 356
memory_heap_t
 Heaps, 258
memory_pool_t, 356
 mp_next, 357
 mp_object_size, 357
 mp_provider, 357
mp_next
 memory_pool_t, 357
mp_object_size
 memory_pool_t, 357
mp_provider
 memory_pool_t, 357
msg_t
 Kernel Types, 52
mutex_t
 Mutexes, 169
Mutexes, 167
 _MUTEX_DATA, 168
 chMtxGetNextMutexS, 177
 chMtxLock, 169
 chMtxLockS, 170
 chMtxObjectInit, 169
 chMtxQueueNotEmptyS, 176
 chMtxTryLock, 171
 chMtxTryLockS, 172
 chMtxUnlock, 173
 chMtxUnlockAll, 175
 chMtxUnlockS, 174
 MUTEX_DECL, 168
 mutex_t, 169

n
 time_measurement_t, 361

n_ctxswc
 kernel_stats_t, 352

n_irq
 kernel_stats_t, 352

NOINLINE
 Kernel Types, 51

NOPRIO
 Scheduler, 77

NORMALPRIO
 Scheduler, 77

name
 testcase, 359

next
 heap_header, 349

OUTPUTQUEUE_DECL
 I/O Queues, 234

offset
 tm_calibration_t, 362

output_queue_t
 I/O Queues, 236

p_ctx
 ch_thread, 329
p_epending
 ch_thread, 331

p_flags
 ch_thread, 329

p_mpool
 ch_thread, 332

p_msg
 ch_thread, 331

p_msgqueue
 ch_thread, 331

p_mtxlist
 ch_thread, 331

p_name
 ch_thread, 329

p_newer
 ch_thread, 329

p_next
 ch_thread, 329
 ch_threads_list, 334
 ch_threads_queue, 335

p_older
 ch_thread, 329

p_preempt
 ch_thread, 330

p_prev
 ch_thread, 329
 ch_threads_queue, 335

p_prio
 ch_thread, 329

p_realprio
 ch_thread, 332

p_refs
 ch_thread, 329

p_state
 ch_thread, 329

p_stats
 ch_thread, 332

p_stklimit
 ch_thread, 329

p_time
 ch_thread, 330

p_u
 ch_thread, 331

p_waiting
 ch_thread, 331

PACKED_VAR
 Kernel Types, 51

PORT_ARCHITECTURE_NAME
 Port Layer, 300

PORT_ARCHITECTURE_XXX
 Port Layer, 299

PORT_ARCHITECTURE_XXX_YYY
 Port Layer, 299

PORT_COMPILER_NAME
 Port Layer, 300

PORT_FAST_IRQ_HANDLER
 Port Layer, 301

PORT_IDLE_THREAD_STACK_SIZE
 Port Layer, 300
PORT_INFO
 Port Layer, 300
PORT_INT_REQUIRED_STACK
 Port Layer, 300
PORT_IRQ_EPILOGUE
 Port Layer, 301
PORT_IRQ_HANDLER
 Port Layer, 301
PORT_IRQ_IS_VALID_KERNEL_PRIORITY
 Port Layer, 301
PORT_IRQ_IS_VALID_PRIORITY
 Port Layer, 301
PORT_IRQ_PROLOGUE
 Port Layer, 301
PORT_SETUP_CONTEXT
 Port Layer, 300
PORT_SUPPORTS_RT
 Port Layer, 300
PORT_THD_FUNCTION
 Kernel Types, 51
PORT_USE_ALT_TIMER
 Port Layer, 300
PORT_WA_SIZE
 Port Layer, 300
panic_msg
 ch_system_debug, 326
patternbmk
 testbmk.c, 409
 testbmk.h, 410
patterndyn
 testdyn.c, 411
 testdyn.h, 411
paternevt
 testevt.c, 412
 testevt.h, 412
patternheap
 testheap.c, 413
 testheap.h, 413
patternmbox
 testmbox.c, 414
 testmbox.h, 414
patternmsg
 testmsg.c, 415
 testmsg.h, 415
patternmtx
 testmtx.c, 416
 testmtx.h, 416
patternqueues
 testqueues.c, 417
 testqueues.h, 418
patternsem
 testsem.c, 418
 testsem.h, 419
patternsyst
 testsys.c, 419
 testsys.h, 420

patternthd
 testthd.c, 420
 testthd.h, 421
ph_next
 pool_header, 357
pool_header, 357
 ph_next, 357
Port Layer, 298
 _port_init, 302
 _port_switch, 302
PORT_ARCHITECTURE_NAME, 300
PORT_ARCHITECTURE_XXX, 299
PORT_ARCHITECTURE_XXX_YYY, 299
PORT_COMPILER_NAME, 300
PORT_FAST_IRQ_HANDLER, 301
PORT_IDLE_THREAD_STACK_SIZE, 300
PORT_INFO, 300
PORT_INT_REQUIRED_STACK, 300
PORT_IRQ_EPILOGUE, 301
PORT_IRQ_HANDLER, 301
PORT_IRQ_IS_VALID_KERNEL_PRIORITY, 301
PORT_IRQ_IS_VALID_PRIORITY, 301
PORT_IRQ_PROLOGUE, 301
PORT_SETUP_CONTEXT, 300
PORT_SUPPORTS_RT, 300
PORT_USE_ALT_TIMER, 300
PORT_WA_SIZE, 300
port_disable, 303
port_enable, 304
port_get_irq_status, 302
port_irq_enabled, 302
port_is_isr_context, 303
port_lock, 303
port_lock_from_isr, 303
port_rt_get_counter_value, 304
port_suspend, 304
port_switch, 301
port_unlock, 303
port_unlock_from_isr, 303
port_wait_for_interrupt, 304
stkalign_t, 302
port_disable
 Port Layer, 303
port_enable
 Port Layer, 304
port_extctx, 358
port_get_irq_status
 Port Layer, 302
port_intctx, 358
port_irq_enabled
 Port Layer, 302
port_is_isr_context
 Port Layer, 303
port_lock
 Port Layer, 303
port_lock_from_isr
 Port Layer, 303
port_rt_get_counter_value

Port Layer, 304
port_suspend
 Port Layer, 304
port_switch
 Port Layer, 301
port_unlock
 Port Layer, 303
port_unlock_from_isr
 Port Layer, 303
port_wait_for_interrupt
 Port Layer, 304

Q_EMPTY
 I/O Queues, 233
Q_FULL
 I/O Queues, 233
Q_OK
 I/O Queues, 233
Q_RESET
 I/O Queues, 233
Q_TIMEOUT
 I/O Queues, 233
q_buffer
 io_queue, 351
q_counter
 io_queue, 351
q_link
 io_queue, 351
q_notify
 io_queue, 351
q_rptr
 io_queue, 351
q_top
 io_queue, 351
q_waiting
 io_queue, 351
q_wptr
 io_queue, 351
qnotify_t
 I/O Queues, 236
queue_dequeue
 Scheduler, 83
queue_fifo_remove
 Scheduler, 83
queue_init
 Scheduler, 90
queue_insert
 Scheduler, 83
queue_isempty
 Scheduler, 92
queue_lifo_remove
 Scheduler, 83
queue_notempty
 Scheduler, 92
queue_prio_insert
 Scheduler, 82

REG_INSERT
 Registry, 279

REG_REMOVE
 Registry, 279

ROMCONST
 Kernel Types, 50

RT Kernel, 37

RTC2MS
 System Management, 59

RTC2S
 System Management, 59

RTC2US
 System Management, 59

rdymsg
 ch_thread, 330

ready_list_t
 Scheduler, 81

Registry, 278
 chRegFirstThread, 279
 chRegGetThreadNameX, 281
 chRegNextThread, 280
 chRegSetThreadName, 281
 chRegSetThreadNameX, 282
 REG_INSERT, 279
 REG_REMOVE, 279

rlist
 ch_system, 324

rtcnt_t
 Kernel Types, 51

rttime_t
 Kernel Types, 51

S2RTC
 System Management, 57

S2ST
 Time and Virtual Timers, 124

s_cnt
 ch_semaphore, 321

s_queue
 ch_semaphore, 321

SEMAPHORE_DECL
 Counting Semaphores, 142

ST2MS
 Time and Virtual Timers, 126

ST2S
 Time and Virtual Timers, 125

ST2US
 Time and Virtual Timers, 126

Scheduler, 73
 _scheduler_init, 82
 ABSPRIO, 77
 CH_FLAG_MODE_HEAP, 79
 CH_FLAG_MODE_MASK, 79
 CH_FLAG_MODE_MPOOL, 79
 CH_FLAG_MODE_STATIC, 79
 CH_FLAG_TERMINATE, 79
 CH_STATE_CURRENT, 78
 CH_STATE_FINAL, 79
 CH_STATE_NAMES, 79
 CH_STATE_QUEUED, 78
 CH_STATE_READY, 77

CH_STATE_SLEEPING, 78
 CH_STATE SNDMSG, 79
 CH_STATE SNDMSGQ, 78
 CH_STATE_SUSPENDED, 78
 CH_STATE_WTANDEVT, 78
 CH_STATE_WTCOND, 78
 CH_STATE_WTEXIT, 78
 CH_STATE_WTMMSG, 79
 CH_STATE_WTMTX, 78
 CH_STATE_WTOREVT, 78
 CH_STATE_WTSEM, 78
 CH_STATE_WTSTART, 78
 ch, 94
 ch_system_t, 82
 chSchCanYieldS, 93
 chSchDoReschedule, 89
 chSchDoRescheduleAhead, 89
 chSchDoRescheduleBehind, 88
 chSchDoYieldS, 93
 chSchGoSleepS, 85
 chSchGoSleepTimeoutS, 86
 chSchIsPreemptionRequired, 88
 chSchIsRescRequiredI, 92
 chSchPreemption, 94
 chSchReadyI, 84
 chSchRescheduleS, 87
 chSchWakeups, 87
 currp, 80
 firstprio, 80
 HIGHPRIO, 77
 IDLEPRIO, 77
 LOWPRIO, 77
 list_init, 90
 list_insert, 84
 list_isempty, 90
 list_notempty, 90
 list_remove, 84
 MSG_OK, 77
 MSG_RESET, 77
 MSG_TIMEOUT, 77
 NOPRIO, 77
 NORMALPRIO, 77
 queue_dequeue, 83
 queue_fifo_remove, 83
 queue_init, 90
 queue_insert, 83
 queue_isempty, 92
 queue_lifo_remove, 83
 queue_notempty, 92
 queue_prio_insert, 82
 ready_list_t, 81
 setcurrp, 81
 system_debug_t, 82
 systime_t, 81
 THD_ALIGN_STACK_SIZE, 79
 THD_FUNCTION, 80
 THD_WORKING_AREA, 80
 THD_WORKING_AREA_SIZE, 80
 thread_reference_t, 81
 thread_t, 81
 threads_list_t, 81
 threads_queue_t, 81
 virtual_timer_t, 81
 virtual_timers_list_t, 82
 vtfunc_t, 81
 se_state
 ch_swc_event_t, 323
 se_time
 ch_swc_event_t, 323
 se_tp
 ch_swc_event_t, 323
 se_wtobjp
 ch_swc_event_t, 323
 semaphore_t
 Counting Semaphores, 143
 setcurrp
 Scheduler, 81
 setup
 testcase, 359
 size
 heap_header, 349
 Statistics, 295
 _stats_ctxswc, 296
 _stats_increase_irq, 295
 _stats_init, 295
 _stats_start_measure_crit_isr, 297
 _stats_start_measure_crit_thd, 296
 _stats_stop_measure_crit_isr, 297
 _stats_stop_measure_crit_thd, 296
 stkalign_t
 Port Layer, 302
 Streams and Files, 274
 Synchronization, 140
 Synchronous Messages, 209
 chMsgGet, 213
 chMsgIsPendingI, 212
 chMsgRelease, 211
 chMsgReleaseS, 213
 chMsgSend, 209
 chMsgWait, 210
 syssts_t
 Kernel Types, 51
 System Management, 54
 _idle_thread, 61
 CH_FAST_IRQ_HANDLER, 57
 CH_IRQ_EPILOGUE, 57
 CH_IRQ_HANDLER, 57
 CH_IRQ_IS_VALID_KERNEL_PRIORITY, 56
 CH_IRQ_IS_VALID_PRIORITY, 56
 CH_IRQ_PROLOGUE, 56
 chSysDisable, 67
 chSysEnable, 68
 chSysGetIdleThreadX, 72
 chSysGetRealtimeCounterX, 60
 chSysGetStatusAndLockX, 64
 chSysHalt, 62

chSysInit, 61
chSysIntegrityCheckI, 63
chSysIsCounterWithinX, 66
chSysLock, 69
chSysLockFromISR, 70
chSysPolledDelayX, 67
chSysRestoreStatusX, 65
chSysSuspend, 68
chSysSwitch, 60
chSysTimerHandlerI, 64
chSysUnconditionalLock, 71
chSysUnconditionalUnlock, 71
chSysUnlock, 69
chSysUnlockFromISR, 70
MS2RTC, 58
RTC2MS, 59
RTC2S, 59
RTC2US, 59
S2RTC, 57
US2RTC, 58
system_debug_t
 Scheduler, 82
system_time_t
 Scheduler, 81

TEST_NO_BENCHMARKS
 Test Runtime, 306
THD_ALIGN_STACK_SIZE
 Scheduler, 79
THD_FUNCTION
 Scheduler, 80
THD_WORKING_AREA
 ch_system, 324
 Scheduler, 80
THD_WORKING_AREA_SIZE
 Scheduler, 80
TIME_IMMEDIATE
 Time and Virtual Timers, 124
TIME_INFINITE
 Time and Virtual Timers, 124
TRUE
 Kernel Types, 50
tb_buffer
 ch_trace_buffer_t, 337
tb_ptr
 ch_trace_buffer_t, 337
tb_size
 ch_trace_buffer_t, 337
teardown
 testcase, 359
Test Runtime, 305
 DELAY_BETWEEN_TESTS, 306
 TEST_NO_BENCHMARKS, 306
 test_assert, 306
 test_assert_lock, 306
 test_assert_sequence, 307
 test_assert_time_window, 307
 test_emit_token, 308
 test_fail, 306
 test_print, 307
 test_println, 308
 test_printn, 307
 test_start_timer, 310
 test_terminate_threads, 308
 test_timer_done, 311
 test_wait_threads, 309
 test_wait_tick, 309
 TestThread, 310
 test.c, 407
 test.h, 408
 test_assert
 Test Runtime, 306
 test_assert_lock
 Test Runtime, 306
 test_assert_sequence
 Test Runtime, 307
 test_assert_time_window
 Test Runtime, 307
 test_emit_token
 Test Runtime, 308
 test_fail
 Test Runtime, 306
 test_print
 Test Runtime, 307
 test_println
 Test Runtime, 308
 test_printn
 Test Runtime, 307
 test_start_timer
 Test Runtime, 310
 test_terminate_threads
 Test Runtime, 308
 test_timer_done
 Test Runtime, 311
 test_wait_threads
 Test Runtime, 309
 test_wait_tick
 Test Runtime, 309
 TestThread
 Test Runtime, 310
 testbmk.c, 409
 patternbmk, 409
 testbmk.h, 410
 patternbmk, 410
 testcase, 359
 execute, 360
 name, 359
 setup, 359
 teardown, 359
 testdyn.c, 410
 patterndyn, 411
 testdyn.h, 411
 patterndyn, 411
 testevt.c, 411
 patternevt, 412
 testevt.h, 412
 patternevt, 412

testheap.c, 412
 patternheap, 413
 testheap.h, 413
 patternheap, 413
 testmbox.c, 413
 patternmbox, 414
 testmbox.h, 414
 patternmbox, 414
 testmsg.c, 414
 patternmsg, 415
 testmsg.h, 415
 patternmsg, 415
 testmtx.c, 415
 patternmtx, 416
 testmtx.h, 416
 patternmtx, 416
 testpools.c, 416
 testpools.h, 417
 testqueues.c, 417
 patternqueues, 417
 testqueues.h, 417
 patternqueues, 418
 testsem.c, 418
 patternsem, 418
 testsem.h, 418
 patternsem, 419
 testsys.c, 419
 patternsyst, 419
 testsys.h, 419
 patternsyst, 420
 testthd.c, 420
 patternthd, 420
 testthd.h, 420
 patternthd, 421
 tfunc_t
 Threads, 98
 thread_reference_t
 Scheduler, 81
 thread_t
 Scheduler, 81
 Threads, 95
 __TREADS_QUEUE_DATA, 97
 __TREADS_QUEUE_DECL, 97
 __thread_init, 98
 __thread_memfill, 99
 chThdCreatel, 99
 chThdCreateStatic, 100
 chThdDequeueAll, 116
 chThdDequeueNextl, 116
 chThdDoDequeueNextl, 121
 chThdEnqueueTimeoutS, 115
 chThdExit, 108
 chThdExitS, 109
 chThdGetPriorityX, 117
 chThdGetSelfX, 117
 chThdGetTicksX, 118
 chThdQueueIsEmptyl, 120
 chThdQueueObjectInit, 120
 chThdResume, 114
 chThdResumel, 113
 chThdResumeS, 113
 chThdSetPriority, 102
 chThdShouldTerminateX, 118
 chThdSleep, 104
 chThdSleepMicroseconds, 98
 chThdSleepMilliseconds, 97
 chThdSleepS, 119
 chThdSleepSeconds, 97
 chThdSleepUntil, 105
 chThdSleepUntilWindowed, 106
 chThdStart, 101
 chThdStartl, 119
 chThdSuspendS, 111
 chThdSuspendTimeoutS, 112
 chThdTerminate, 103
 chThdTerminatedX, 118
 chThdWait, 110
 chThdYield, 107
 tfunc_t, 98
 threads_list_t
 Scheduler, 81
 threads_queue_t
 Scheduler, 81
 Time and Virtual Timers, 123
 __vt_init, 127
 chVTDoResetl, 128
 chVTDoSetl, 127
 chVTDoTickl, 138
 chVTGetSystemTime, 129
 chVTGetSystemTimeX, 129
 chVTGetTimersStatel, 132
 chVTIsArmed, 133
 chVTIsArmedl, 133
 chVTIsSystemTimeWithin, 131
 chVTIsSystemTimeWithinX, 131
 chVTIsTimeWithinX, 130
 chVTOBJECTINIT, 128
 chVTReset, 135
 chVTResetl, 134
 chVTSet, 137
 chVTSetl, 136
 chVTTimeElapsedSinceX, 130
 MS2ST, 124
 S2ST, 124
 ST2MS, 126
 ST2S, 125
 ST2US, 126
 TIME_IMMEDIATE, 124
 TIME_INFINITE, 124
 US2ST, 125
 Time Measurement, 292
 __tm_init, 292
 chTMChainMeasurementToX, 294
 chTMOBJECTINIT, 292
 chTMStartMeasurementX, 294
 chTMStopMeasurementX, 294

time_measurement_t, 360
 best, 361
 cumulative, 361
 last, 361
 n, 361
 worst, 361

tm
 ch_system, 324

tm_calibration_t, 361
 offset, 362

tmode_t
 Kernel Types, 51

tprio_t
 Kernel Types, 52

trace_buffer
 ch_system_debug, 326

tresfs_t
 Kernel Types, 51

tslices_t
 Kernel Types, 51

tstate_t
 Kernel Types, 51

u
 heap_header, 349

US2RTC
 System Management, 58

US2ST
 Time and Virtual Timers, 125

ucnt_t
 Kernel Types, 52

Version Numbers and Identification, 38
 _CHIBIOS_RT_, 38
 CH_KERNEL_MAJOR, 38
 CH_KERNEL_MINOR, 38
 CH_KERNEL_PATCH, 38
 CH_KERNEL_STABLE, 38
 CH_KERNEL_VERSION, 38

virtual_timer_t
 Scheduler, 81

virtual_timers_list_t
 Scheduler, 82

vmt
 BaseSequentialStream, 316

vt_delta
 ch_virtual_timer, 338
 ch_virtual_timers_list, 341

vt_func
 ch_virtual_timer, 339

vt_lasttime
 ch_virtual_timers_list, 341

vt_next
 ch_virtual_timer, 338
 ch_virtual_timers_list, 340

vt_par
 ch_virtual_timer, 339

vt_prev
 ch_virtual_timer, 338
 ch_virtual_timers_list, 340

ch_virtual_timers_list, 340

vt_systime
 ch_virtual_timers_list, 341

vtfunc_t
 Scheduler, 81

vtlist
 ch_system, 324

worst
 time_measurement_t, 361

wtmtp
 ch_thread, 331

wtobjp
 ch_thread, 330

wtsemp
 ch_thread, 330

wtrp
 ch_thread, 330