

ChibiOS/RT 2.6.9

Kernel and HAL Reference Manual

Contents

1 ChibiOS/RT	1
1.1 Copyright	1
1.2 Introduction	1
1.3 Related Documents	1
2 Kernel Concepts	2
2.1 Naming Conventions	2
2.2 API Name Suffixes	2
2.3 Interrupt Classes	3
2.4 System States	3
2.5 Scheduling	5
2.6 Thread States	6
2.7 Priority Levels	6
2.8 Thread Working Area	6
3 Testing Strategy	8
3.1 Threads and Scheduler test	9
3.1.1 Ready List functionality #1	9
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	10
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	11
3.3.1 Messages Server loop	12
3.4 Semaphores test	12
3.4.1 Enqueuing test	12
3.4.2 Timeout test	12
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	14
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	15
3.5.8	Condition Variable priority boost test	16
3.6	Events test	16
3.6.1	Events registration and dispatch	16
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	18
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	Kernel Benchmarks	20
3.11.1	Messages performance #1	21
3.11.2	Messages performance #2	21
3.11.3	Messages performance #3	21
3.11.4	Context Switch performance	21
3.11.5	Threads performance, full cycle	21
3.11.6	Threads performance, create/exit only	22
3.11.7	Mass reschedule performance	22
3.11.8	I/O Round-Robin voluntary reschedule.	22
3.11.9	I/O Queues throughput	22
3.11.10	Virtual Timers set/reset performance	22
3.11.11	Semaphores wait/signal performance	22
3.11.12	Mutexes lock/unlock performance	22
3.11.13	RAM Footprint	23
4	Deprecated List	24
5	Module Index	25
5.1	Modules	25

6 Namespace Index	27
6.1 Namespace List	27
7 Class Hierarchy Index	28
7.1 Class Hierarchy	28
8 Data Structure Index	31
8.1 Data Structures	31
9 File Index	34
9.1 File List	34
10 Module Documentation	37
10.1 Kernel	37
10.1.1 Detailed Description	37
10.2 Version Numbers and Identification	37
10.2.1 Detailed Description	37
10.2.2 Function Documentation	38
10.2.2.1 _idle_thread	38
10.2.3 Define Documentation	38
10.2.3.1 _CHIBIOS_RT_	38
10.2.3.2 CH_KERNEL_VERSION	39
10.2.3.3 CH_KERNEL_MAJOR	39
10.2.3.4 CH_KERNEL_MINOR	39
10.2.3.5 CH_KERNEL_PATCH	39
10.2.3.6 FALSE	39
10.2.3.7 TRUE	39
10.2.3.8 CH_SUCCESS	39
10.2.3.9 CH_FAILED	39
10.3 Configuration	39
10.3.1 Detailed Description	39
10.3.2 Define Documentation	41
10.3.2.1 CH_FREQUENCY	41
10.3.2.2 CH_TIME_QUANTUM	41
10.3.2.3 CH_MEMCORE_SIZE	42
10.3.2.4 CH_NO_IDLE_THREAD	42
10.3.2.5 CH_OPTIMIZE_SPEED	42
10.3.2.6 CH_USE_REGISTRY	42
10.3.2.7 CH_USE_WAITEXIT	42
10.3.2.8 CH_USE_SEMAPHORES	43
10.3.2.9 CH_USE_SEMAPHORES_PRIORITY	43
10.3.2.10 CH_USE_SEMSW	43

10.3.2.11 CH_USE_MUTEXES	43
10.3.2.12 CH_USE_CONDVAR	43
10.3.2.13 CH_USE_CONDVAR_TIMEOUT	44
10.3.2.14 CH_USE_EVENTS	44
10.3.2.15 CH_USE_EVENTS_TIMEOUT	44
10.3.2.16 CH_USE_MESSAGES	44
10.3.2.17 CH_USE_MESSAGES_PRIORITY	44
10.3.2.18 CH_USE_MAILBOXES	45
10.3.2.19 CH_USE_QUEUES	45
10.3.2.20 CH_USE_MEMCORE	45
10.3.2.21 CH_USE_HEAP	45
10.3.2.22 CH_USE_MALLOC_HEAP	45
10.3.2.23 CH_USE_MEMPOOLS	46
10.3.2.24 CH_USE_DYNAMIC	46
10.3.2.25 CH_DBG_SYSTEM_STATE_CHECK	46
10.3.2.26 CH_DBG_ENABLE_CHECKS	46
10.3.2.27 CH_DBG_ENABLE_ASSERTS	46
10.3.2.28 CH_DBG_ENABLE_TRACE	47
10.3.2.29 CH_DBG_ENABLE_STACK_CHECK	47
10.3.2.30 CH_DBG_FILL_THREADS	47
10.3.2.31 CH_DBG_THREADS_PROFILING	47
10.3.2.32 THREAD_EXT_FIELDS	47
10.3.2.33 THREAD_EXT_INIT_HOOK	47
10.3.2.34 THREAD_EXT_EXIT_HOOK	48
10.3.2.35 THREAD_CONTEXT_SWITCH_HOOK	48
10.3.2.36 IDLE_LOOP_HOOK	48
10.3.2.37 SYSTEM_TICK_EVENT_HOOK	48
10.3.2.38 SYSTEM_HALT_HOOK	49
10.4 Types	49
10.4.1 Detailed Description	49
10.4.2 Define Documentation	50
10.4.2.1 INLINE	50
10.4.2.2 ROMCONST	50
10.4.2.3 PACK_STRUCT_STRUCT	50
10.4.2.4 PACK_STRUCT_BEGIN	50
10.4.2.5 PACK_STRUCT_END	50
10.4.3 Typedef Documentation	50
10.4.3.1 bool_t	50
10.4.3.2 tmode_t	50
10.4.3.3 tstate_t	50

10.4.3.4	<code>trefs_t</code>	50
10.4.3.5	<code>tprio_t</code>	50
10.4.3.6	<code>msg_t</code>	51
10.4.3.7	<code>eventid_t</code>	51
10.4.3.8	<code>eventmask_t</code>	51
10.4.3.9	<code>systime_t</code>	51
10.4.3.10	<code>cnt_t</code>	51
10.5	Base Kernel Services	51
10.5.1	Detailed Description	51
10.6	System Management	51
10.6.1	Detailed Description	51
10.6.2	Function Documentation	53
10.6.2.1	<code>chSysInit</code>	53
10.6.2.2	<code>chSysTimerHandlerI</code>	53
10.6.2.3	<code>WORKING_AREA</code>	54
10.6.2.4	<code>_idle_thread</code>	54
10.6.3	Define Documentation	54
10.6.3.1	<code>chSysGetIdleThread</code>	54
10.6.3.2	<code>chSysHalt</code>	55
10.6.3.3	<code>chSysSwitch</code>	55
10.6.3.4	<code>chSysDisable</code>	55
10.6.3.5	<code>chSysSuspend</code>	56
10.6.3.6	<code>chSysEnable</code>	56
10.6.3.7	<code>chSysLock</code>	57
10.6.3.8	<code>chSysUnlock</code>	57
10.6.3.9	<code>chSysLockFromIlsr</code>	57
10.6.3.10	<code>chSysUnlockFromIlsr</code>	57
10.6.3.11	<code>CH_IRQ_PROLOGUE</code>	58
10.6.3.12	<code>CH_IRQ_EPILOGUE</code>	58
10.6.3.13	<code>CH_IRQ_HANDLER</code>	58
10.6.3.14	<code>CH_FAST_IRQ_HANDLER</code>	59
10.7	Scheduler	59
10.7.1	Detailed Description	59
10.7.2	Function Documentation	61
10.7.2.1	<code>_scheduler_init</code>	61
10.7.2.2	<code>chSchReadyI</code>	61
10.7.2.3	<code>chSchGoSleepS</code>	61
10.7.2.4	<code>chSchGoSleepTimeoutS</code>	62
10.7.2.5	<code>chSchWakeupS</code>	63
10.7.2.6	<code>chSchRescheduleS</code>	63

10.7.2.7	chSchIsPreemptionRequired	64
10.7.2.8	chSchDoRescheduleBehind	64
10.7.2.9	chSchDoRescheduleAhead	65
10.7.2.10	chSchDoReschedule	65
10.7.3	Variable Documentation	66
10.7.3.1	rlist	66
10.7.4	Define Documentation	66
10.7.4.1	RDY_OK	66
10.7.4.2	RDY_TIMEOUT	66
10.7.4.3	RDY_RESET	66
10.7.4.4	NOPRIO	66
10.7.4.5	IDLEPRIO	66
10.7.4.6	LOWPRIO	66
10.7.4.7	NORMALPRIO	66
10.7.4.8	HIGHPRIO	67
10.7.4.9	ABSPRIO	67
10.7.4.10	TIME_IMMEDIATE	67
10.7.4.11	TIME_INFINITE	67
10.7.4.12	firstprio	67
10.7.4.13	currp	67
10.7.4.14	setcurrp	67
10.7.4.15	chSchIsRescRequired	68
10.7.4.16	chSchCanYieldS	68
10.7.4.17	chSchDoYieldS	68
10.7.4.18	chSchPreemption	68
10.8	Threads	69
10.8.1	Detailed Description	69
10.8.2	Function Documentation	71
10.8.2.1	_thread_init	71
10.8.2.2	_thread_memfill	72
10.8.2.3	chThdCreateL	72
10.8.2.4	chThdCreateStatic	73
10.8.2.5	chThdSetPriority	74
10.8.2.6	chThdResume	74
10.8.2.7	chThdTerminate	75
10.8.2.8	chThdSleep	75
10.8.2.9	chThdSleepUntil	75
10.8.2.10	chThdYield	76
10.8.2.11	chThdExit	76
10.8.2.12	chThdExitS	76

10.8.2.13 chThdWait	77
10.8.3 Define Documentation	78
10.8.3.1 THD_STATE_READY	78
10.8.3.2 THD_STATE_CURRENT	78
10.8.3.3 THD_STATE_SUSPENDED	78
10.8.3.4 THD_STATE_WTSEM	78
10.8.3.5 THD_STATE_WTMXT	79
10.8.3.6 THD_STATE_WTCOND	79
10.8.3.7 THD_STATE_SLEEPING	79
10.8.3.8 THD_STATE_WTEXIT	79
10.8.3.9 THD_STATE_WTOREVT	79
10.8.3.10 THD_STATE_WTANDEV	79
10.8.3.11 THD_STATE_SNDMSGQ	79
10.8.3.12 THD_STATE_SNDMSG	79
10.8.3.13 THD_STATE_WTMSG	79
10.8.3.14 THD_STATE_WTQUEUE	79
10.8.3.15 THD_STATE_FINAL	79
10.8.3.16 THD_STATE_NAMES	79
10.8.3.17 THD_MEM_MODE_MASK	80
10.8.3.18 THD_MEM_MODE_STATIC	80
10.8.3.19 THD_MEM_MODE_HEAP	80
10.8.3.20 THD_MEM_MODE_MEMPOOL	80
10.8.3.21 THD_TERMINATE	80
10.8.3.22 chThdSelf	80
10.8.3.23 chThdGetPriority	80
10.8.3.24 chThdGetTicks	80
10.8.3.25 chThdLS	81
10.8.3.26 chThdTerminated	81
10.8.3.27 chThdShouldTerminate	81
10.8.3.28 chThdResumeL	82
10.8.3.29 chThdSleepS	82
10.8.3.30 chThdSleepSeconds	82
10.8.3.31 chThdSleepMilliseconds	82
10.8.3.32 chThdSleepMicroseconds	83
10.8.4 Typedef Documentation	83
10.8.4.1 tfunc_t	83
10.9 Time and Virtual Timers	83
10.9.1 Detailed Description	83
10.9.2 Function Documentation	84
10.9.2.1 _vt_init	84

10.9.2.2 chVTSetl	85
10.9.2.3 chVTResetl	85
10.9.3 Variable Documentation	86
10.9.3.1 vtlist	86
10.9.3.2 vtlist	86
10.9.4 Define Documentation	86
10.9.4.1 S2ST	86
10.9.4.2 MS2ST	86
10.9.4.3 US2ST	87
10.9.4.4 chVTDoTickl	87
10.9.4.5 chVTIsArmedl	88
10.9.4.6 chVTSet	88
10.9.4.7 chVTReset	88
10.9.4.8 chTimeNow	89
10.9.4.9 chTimeElapsedSince	89
10.9.4.10 chTimelsWithin	89
10.9.5 Typedef Documentation	90
10.9.5.1 vtfunc_t	90
10.9.5.2 VirtualTimer	90
10.10 Synchronization	90
10.10.1 Detailed Description	90
10.11 Counting Semaphores	90
10.11.1 Detailed Description	90
10.11.2 Function Documentation	92
10.11.2.1 chSemInit	92
10.11.2.2 chSemReset	92
10.11.2.3 chSemResetl	93
10.11.2.4 chSemWait	94
10.11.2.5 chSemWaitS	94
10.11.2.6 chSemWaitTimeout	95
10.11.2.7 chSemWaitTimeoutS	96
10.11.2.8 chSemSignal	96
10.11.2.9 chSemSignall	97
10.11.2.10 chSemAddCounterl	97
10.11.2.11 chSemSignalWait	98
10.11.3 Define Documentation	99
10.11.3.1 _SEMAPHORE_DATA	99
10.11.3.2 SEMAPHORE_DECL	99
10.11.3.3 chSemFastWaitl	99
10.11.3.4 chSemFastSignall	99

10.11.3.5 chSemGetCounterl	100
10.11.4 Typedef Documentation	100
10.11.4.1 Semaphore	100
10.12 Binary Semaphores	100
10.12.1 Detailed Description	100
10.12.2 Define Documentation	101
10.12.2.1 _BSEMAPHORE_DATA	101
10.12.2.2 BSEMAPHORE_DECL	101
10.12.2.3 chBSemInit	102
10.12.2.4 chBSemWait	102
10.12.2.5 chBSemWaitS	102
10.12.2.6 chBSemWaitTimeout	103
10.12.2.7 chBSemWaitTimeoutS	103
10.12.2.8 chBSemReset	103
10.12.2.9 chBSemResetl	104
10.12.2.10 chBSemSignal	104
10.12.2.11 chBSemSignall	105
10.12.2.12 chBSemGetStatel	105
10.13 Mutexes	105
10.13.1 Detailed Description	105
10.13.2 Function Documentation	107
10.13.2.1 chMtxInit	107
10.13.2.2 chMtxLock	107
10.13.2.3 chMtxLockS	108
10.13.2.4 chMtxTryLock	109
10.13.2.5 chMtxTryLockS	109
10.13.2.6 chMtxUnlock	110
10.13.2.7 chMtxUnlockS	111
10.13.2.8 chMtxUnlockAll	111
10.13.3 Define Documentation	112
10.13.3.1 _MUTEX_DATA	112
10.13.3.2 MUTEX_DECL	112
10.13.3.3 chMtxQueueNotEmptyS	112
10.13.4 Typedef Documentation	113
10.13.4.1 Mutex	113
10.14 Condition Variables	113
10.14.1 Detailed Description	113
10.14.2 Function Documentation	114
10.14.2.1 chCondInit	114
10.14.2.2 chCondSignal	114

10.14.2.3 chCondSignal	115
10.14.2.4 chCondBroadcast	115
10.14.2.5 chCondBroadcastl	116
10.14.2.6 chCondWait	116
10.14.2.7 chCondWaitS	117
10.14.2.8 chCondWaitTimeout	118
10.14.2.9 chCondWaitTimeoutS	119
10.14.3 Define Documentation	120
10.14.3.1 _CONDVAR_DATA	120
10.14.3.2 CONDVAR_DECL	120
10.14.4 Ttypedef Documentation	120
10.14.4.1 CondVar	120
10.15Event Flags	120
10.15.1 Detailed Description	121
10.15.2 Function Documentation	123
10.15.2.1 chEvtRegisterMask	123
10.15.2.2 chEvtUnregister	123
10.15.2.3 chEvtGetAndClearEvents	123
10.15.2.4 chEvtAddEvents	124
10.15.2.5 chEvtGetAndClearFlags	124
10.15.2.6 chEvtGetAndClearFlagsl	124
10.15.2.7 chEvtSignal	125
10.15.2.8 chEvtSignall	125
10.15.2.9 chEvtBroadcastFlags	126
10.15.2.10chEvtBroadcastFlagsl	126
10.15.2.11chEvtDispatch	127
10.15.2.12chEvtWaitOneTimeout	127
10.15.2.13chEvtWaitAnyTimeout	128
10.15.2.14chEvtWaitAllTimeout	129
10.15.2.15chEvtWaitOne	130
10.15.2.16chEvtWaitAny	130
10.15.2.17chEvtWaitAll	131
10.15.3 Define Documentation	131
10.15.3.1 _EVENTSOURCE_DATA	131
10.15.3.2 EVENTSOURCE_DECL	132
10.15.3.3 ALL_EVENTS	132
10.15.3.4 EVENT_MASK	132
10.15.3.5 chEvtRegister	132
10.15.3.6 chEvtInit	132
10.15.3.7 chEvtIsListeningl	133

10.15.3.8 chEvtBroadcast	133
10.15.3.9 chEvtBroadcastl	133
10.15.4 Ttypedef Documentation	133
10.15.4.1 EventSource	133
10.15.4.2 evhandler_t	133
10.16 Synchronous Messages	133
10.16.1 Detailed Description	134
10.16.2 Function Documentation	134
10.16.2.1 chMsgSend	134
10.16.2.2 chMsgWait	135
10.16.2.3 chMsgRelease	136
10.16.3 Define Documentation	136
10.16.3.1 chMsgIsPending	136
10.16.3.2 chMsgGet	136
10.16.3.3 chMsgReleaseS	136
10.17 Mailboxes	137
10.17.1 Detailed Description	137
10.17.2 Function Documentation	138
10.17.2.1 chMBInit	138
10.17.2.2 chMBReset	139
10.17.2.3 chMBPost	139
10.17.2.4 chMBPostS	140
10.17.2.5 chMBPostl	141
10.17.2.6 chMBPostAhead	142
10.17.2.7 chMBPostAheadS	142
10.17.2.8 chMBPostAheadl	143
10.17.2.9 chMBFetch	144
10.17.2.10 chMBFetchS	145
10.17.2.11 chMBFetchl	145
10.17.3 Define Documentation	146
10.17.3.1 chMBSizel	146
10.17.3.2 chMBGetFreeCountl	146
10.17.3.3 chMBGetUsedCountl	147
10.17.3.4 chMBPeekl	147
10.17.3.5 _MAILBOX_DATA	147
10.17.3.6 MAILBOX_DECL	148
10.18 I/O Queues	148
10.18.1 Detailed Description	148
10.18.2 Function Documentation	150
10.18.2.1 chIQInit	150

10.18.2.2 chIQReset	150
10.18.2.3 chIQPut	151
10.18.2.4 chIQGetTimeout	152
10.18.2.5 chIQReadTimeout	152
10.18.2.6 chOQInit	153
10.18.2.7 chOQReset	153
10.18.2.8 chOQPutTimeout	154
10.18.2.9 chOQGet	154
10.18.2.10 chOQWriteTimeout	155
10.18.3 Define Documentation	156
10.18.3.1 Q_OK	156
10.18.3.2 Q_TIMEOUT	156
10.18.3.3 Q_RESET	156
10.18.3.4 Q_EMPTY	156
10.18.3.5 Q_FULL	156
10.18.3.6 chQSizel	156
10.18.3.7 chQSpaceI	156
10.18.3.8 chQGetLink	157
10.18.3.9 chIQGetFull	157
10.18.3.10 chIQGetEmpty	157
10.18.3.11 chIQIsEmpty	158
10.18.3.12 chIQIsFull	158
10.18.3.13 chIQGet	158
10.18.3.14 INPUTQUEUE_DATA	159
10.18.3.15 INPUTQUEUE_DECL	159
10.18.3.16 chOQGetFull	159
10.18.3.17 chOQGetEmpty	160
10.18.3.18 chOQIsEmpty	160
10.18.3.19 chOQIsFull	161
10.18.3.20 chOQPut	161
10.18.3.21 OUTPUTQUEUE_DATA	161
10.18.3.22 OUTPUTQUEUE_DECL	162
10.18.4 Typedef Documentation	162
10.18.4.1 GenericQueue	162
10.18.4.2 qnotify_t	162
10.18.4.3 InputQueue	162
10.18.4.4 OutputQueue	162
10.19 Memory Management	163
10.19.1 Detailed Description	163
10.20 Core Memory Manager	163

10.20.1 Detailed Description	163
10.20.2 Function Documentation	164
10.20.2.1 <code>_core_init</code>	164
10.20.2.2 <code>chCoreAlloc</code>	164
10.20.2.3 <code>chCoreAlloc1</code>	165
10.20.2.4 <code>chCoreStatus</code>	165
10.20.3 Define Documentation	166
10.20.3.1 <code>MEM_ALIGN_SIZE</code>	166
10.20.3.2 <code>MEM_ALIGN_MASK</code>	166
10.20.3.3 <code>MEM_ALIGN_PREV</code>	166
10.20.3.4 <code>MEM_ALIGN_NEXT</code>	166
10.20.3.5 <code>MEM_IS_ALIGNED</code>	166
10.20.4 Typedef Documentation	166
10.20.4.1 <code>memgetfunc_t</code>	166
10.21 Heps	166
10.21.1 Detailed Description	166
10.21.2 Function Documentation	167
10.21.2.1 <code>_heap_init</code>	167
10.21.2.2 <code>chHeapInit</code>	167
10.21.2.3 <code>chHeapAlloc</code>	168
10.21.2.4 <code>chHeapFree</code>	168
10.21.2.5 <code>chHeapStatus</code>	169
10.22 Memory Pools	169
10.22.1 Detailed Description	169
10.22.2 Function Documentation	170
10.22.2.1 <code>chPoolInit</code>	170
10.22.2.2 <code>chPoolLoadArray</code>	170
10.22.2.3 <code>chPoolAlloc1</code>	171
10.22.2.4 <code>chPoolAlloc</code>	171
10.22.2.5 <code>chPoolFree1</code>	172
10.22.2.6 <code>chPoolFree</code>	173
10.22.3 Define Documentation	173
10.22.3.1 <code>_MEMORYPOOL_DATA</code>	173
10.22.3.2 <code>MEMORYPOOL_DECL</code>	174
10.22.3.3 <code>chPoolAdd</code>	174
10.22.3.4 <code>chPoolAdd1</code>	174
10.23 Dynamic Threads	175
10.23.1 Detailed Description	175
10.23.2 Function Documentation	175
10.23.2.1 <code>chThdAddRef</code>	175

10.23.2.2 chThdRelease	175
10.23.2.3 chThdCreateFromHeap	176
10.23.2.4 chThdCreateFromMemoryPool	177
10.24 Streams and Files	178
10.24.1 Detailed Description	178
10.25 Abstract Sequential Streams	178
10.25.1 Detailed Description	178
10.25.2 Define Documentation	179
10.25.2.1 <code>_base_sequential_stream_methods</code>	179
10.25.2.2 <code>_base_sequential_stream_data</code>	179
10.25.2.3 chSequentialStreamWrite	179
10.25.2.4 chSequentialStreamRead	180
10.25.2.5 chSequentialStreamPut	180
10.25.2.6 chSequentialStreamGet	180
10.26 Abstract File Streams	181
10.26.1 Detailed Description	181
10.26.2 Define Documentation	182
10.26.2.1 FILE_OK	182
10.26.2.2 FILE_ERROR	182
10.26.2.3 <code>_base_file_stream_methods</code>	182
10.26.2.4 <code>_base_file_stream_data</code>	182
10.26.2.5 chFileStreamClose	182
10.26.2.6 chFileStreamGetError	183
10.26.2.7 chFileStreamGetSize	183
10.26.2.8 chFileStreamGetPosition	183
10.26.2.9 chFileStreamSeek	183
10.26.3 Typedef Documentation	184
10.26.3.1 <code>fileoffset_t</code>	184
10.27 Registry	184
10.27.1 Detailed Description	184
10.27.2 Function Documentation	185
10.27.2.1 chRegFirstThread	185
10.27.2.2 chRegNextThread	185
10.27.3 Define Documentation	186
10.27.3.1 chRegSetThreadName	186
10.27.3.2 chRegGetThreadName	186
10.27.3.3 REG_REMOVE	186
10.27.3.4 REG_INSERT	187
10.28 Debug	187
10.28.1 Detailed Description	187

10.28.2 Function Documentation	189
10.28.2.1 _trace_init	189
10.28.2.2 dbg_trace	189
10.28.2.3 dbg_check_disable	189
10.28.2.4 dbg_check_suspend	190
10.28.2.5 dbg_check_enable	190
10.28.2.6 dbg_check_lock	191
10.28.2.7 dbg_check_unlock	191
10.28.2.8 dbg_check_lock_from_isr	191
10.28.2.9 dbg_check_unlock_from_isr	192
10.28.2.10 dbg_check_enter_isr	192
10.28.2.11 dbg_check_leave_isr	193
10.28.2.12 chDbgCheckClassI	193
10.28.2.13 chDbgCheckClassS	193
10.28.2.14 chDbgPanic	194
10.28.3 Variable Documentation	194
10.28.3.1 dbg_isr_cnt	194
10.28.3.2 dbg_lock_cnt	194
10.28.3.3 dbg_trace_buffer	194
10.28.3.4 dbg_panic_msg	194
10.28.4 Define Documentation	194
10.28.4.1 CH_TRACE_BUFFER_SIZE	194
10.28.4.2 CH_STACK_FILL_VALUE	194
10.28.4.3 CH_THREAD_FILL_VALUE	195
10.28.4.4 chDbgCheck	195
10.28.4.5 chDbgAssert	195
10.29 Internals	196
10.29.1 Detailed Description	196
10.29.2 Function Documentation	197
10.29.2.1 prio_insert	197
10.29.2.2 queue_insert	197
10.29.2.3 fifo_remove	197
10.29.2.4 lifo_remove	197
10.29.2.5 dequeue	198
10.29.2.6 list_insert	198
10.29.2.7 list_remove	198
10.29.3 Define Documentation	199
10.29.3.1 queue_init	199
10.29.3.2 list_init	199
10.29.3.3 isempty	199

10.29.3.4 <code>notempty</code>	199
10.29.3.5 <code>_THREADSQUEUE_DATA</code>	199
10.29.3.6 <code>THREADSQUEUE_DECL</code>	199
10.30 Port	200
10.30.1 Detailed Description	200
10.30.2 Function Documentation	201
10.30.2.1 <code>port_init</code>	201
10.30.2.2 <code>port_lock</code>	201
10.30.2.3 <code>port_unlock</code>	201
10.30.2.4 <code>port_lock_from_isr</code>	202
10.30.2.5 <code>port_unlock_from_isr</code>	202
10.30.2.6 <code>port_disable</code>	202
10.30.2.7 <code>port_suspend</code>	202
10.30.2.8 <code>port_enable</code>	202
10.30.2.9 <code>port_wait_for_interrupt</code>	202
10.30.2.10 <code>port_halt</code>	202
10.30.2.11 <code>port_switch</code>	203
10.30.3 Define Documentation	203
10.30.3.1 <code>PORT_IDLE_THREAD_STACK_SIZE</code>	203
10.30.3.2 <code>PORT_INT_REQUIRED_STACK</code>	203
10.30.3.3 <code>CH_ARCHITECTURE_XXX</code>	203
10.30.3.4 <code>CH_ARCHITECTURE_NAME</code>	203
10.30.3.5 <code>CH_ARCHITECTURE_VARIANT_NAME</code>	204
10.30.3.6 <code>CH_COMPILER_NAME</code>	204
10.30.3.7 <code>CH_PORT_INFO</code>	204
10.30.3.8 <code>SETUP_CONTEXT</code>	204
10.30.3.9 <code>STACK_ALIGN</code>	204
10.30.3.10 <code>THD_WA_SIZE</code>	204
10.30.3.11 <code>WORKING_AREA</code>	204
10.30.3.12 <code>PORT_IRQ_PROLOGUE</code>	204
10.30.3.13 <code>PORT_IRQ_EPILOGUE</code>	204
10.30.3.14 <code>PORT_IRQ_HANDLER</code>	205
10.30.3.15 <code>PORT_FAST_IRQ_HANDLER</code>	205
10.30.4 Typedef Documentation	205
10.30.4.1 <code>stkalign_t</code>	205
10.31 ADC Driver	205
10.31.1 Detailed Description	205
10.31.2 Driver State Machine	205
10.31.3 ADC Operations	206
10.31.3.1 ADC Conversion Groups	206

10.31.3.2 ADC Conversion Modes	206
10.31.3.3 ADC Callbacks	206
10.31.4 Function Documentation	209
10.31.4.1 adcInit	209
10.31.4.2 adcObjectInit	209
10.31.4.3 adcStart	210
10.31.4.4 adcStop	210
10.31.4.5 adcStartConversion	211
10.31.4.6 adcStartConversionl	211
10.31.4.7 adcStopConversion	212
10.31.4.8 adcStopConversionl	213
10.31.4.9 adcAcquireBus	213
10.31.4.10adcReleaseBus	214
10.31.4.11adcConvert	214
10.31.4.12adc_lld_init	215
10.31.4.13adc_lld_start	216
10.31.4.14adc_lld_stop	216
10.31.4.15adc_lld_start_conversion	216
10.31.4.16adc_lld_stop_conversion	216
10.31.5 Variable Documentation	217
10.31.5.1 ADCD1	217
10.31.6 Define Documentation	217
10.31.6.1 ADC_USE_WAIT	217
10.31.6.2 ADC_USE_MUTUAL_EXCLUSION	217
10.31.6.3 _adc_reset_i	217
10.31.6.4 _adc_reset_s	217
10.31.6.5 _adc_wakeup_isr	218
10.31.6.6 _adc_timeout_isr	218
10.31.6.7 _adc_isr_half_code	219
10.31.6.8 _adc_isr_full_code	219
10.31.6.9 _adc_isr_error_code	220
10.31.6.10PLATFORM_ADC_USE_ADC1	220
10.31.7 Typedef Documentation	220
10.31.7.1 adcsample_t	220
10.31.7.2 adc_channels_num_t	220
10.31.7.3 ADCDriver	220
10.31.7.4 adccallback_t	221
10.31.7.5 adcerrorcallback_t	221
10.31.8 Enumeration Type Documentation	221
10.31.8.1 adcstate_t	221

10.31.8.2 <code>adcerror_t</code>	221
10.32 CAN Driver	221
10.32.1 Detailed Description	221
10.32.2 Driver State Machine	222
10.32.3 Function Documentation	224
10.32.3.1 <code>canInit</code>	224
10.32.3.2 <code>canObjectInit</code>	225
10.32.3.3 <code>canStart</code>	225
10.32.3.4 <code>canStop</code>	226
10.32.3.5 <code>canTransmit</code>	226
10.32.3.6 <code>canReceive</code>	227
10.32.3.7 <code>canSleep</code>	228
10.32.3.8 <code>canWakeup</code>	229
10.32.3.9 <code>can_lld_init</code>	229
10.32.3.10 <code>can_lld_start</code>	230
10.32.3.11 <code>can_lld_stop</code>	230
10.32.3.12 <code>can_lld_is_tx_empty</code>	230
10.32.3.13 <code>can_lld_transmit</code>	231
10.32.3.14 <code>can_lld_is_rx_nonempty</code>	231
10.32.3.15 <code>can_lld_receive</code>	231
10.32.3.16 <code>can_lld_sleep</code>	231
10.32.3.17 <code>can_lld_wakeup</code>	232
10.32.4 Variable Documentation	232
10.32.4.1 <code>CAND1</code>	232
10.32.5 Define Documentation	232
10.32.5.1 <code>CAN_LIMIT_WARNING</code>	232
10.32.5.2 <code>CAN_LIMIT_ERROR</code>	232
10.32.5.3 <code>CAN_BUS_OFF_ERROR</code>	232
10.32.5.4 <code>CAN_FRAMING_ERROR</code>	232
10.32.5.5 <code>CAN_OVERFLOW_ERROR</code>	232
10.32.5.6 <code>CAN_ANY_MAILBOX</code>	232
10.32.5.7 <code>CAN_USE_SLEEP_MODE</code>	232
10.32.5.8 <code>CAN_MAILBOX_TO_MASK</code>	233
10.32.5.9 <code>CAN_SUPPORTS_SLEEP</code>	233
10.32.5.10 <code>CAN_TX_MAILBOXES</code>	233
10.32.5.11 <code>CAN_RX_MAILBOXES</code>	233
10.32.5.12 <code>PLATFORM_CAN_USE_CAN1</code>	233
10.32.6 Typedef Documentation	233
10.32.6.1 <code>canmbx_t</code>	233
10.32.7 Enumeration Type Documentation	233

10.32.7.1 canstate_t	233
10.33EXT Driver	233
10.33.1 Detailed Description	233
10.33.2 Driver State Machine	234
10.33.3 EXT Operations.	234
10.33.4 Function Documentation	236
10.33.4.1 extInit	236
10.33.4.2 extObjectInit	237
10.33.4.3 extStart	237
10.33.4.4 extStop	237
10.33.4.5 extChannelEnable	238
10.33.4.6 extChannelDisable	238
10.33.4.7 extSetChannelModel	238
10.33.4.8 ext_lld_init	239
10.33.4.9 ext_lld_start	239
10.33.4.10ext_lld_stop	240
10.33.4.11ext_lld_channel_enable	240
10.33.4.12ext_lld_channel_disable	240
10.33.5 Variable Documentation	240
10.33.5.1 EXTD1	240
10.33.6 Define Documentation	240
10.33.6.1 EXT_CH_MODE_EDGES_MASK	240
10.33.6.2 EXT_CH_MODE_DISABLED	240
10.33.6.3 EXT_CH_MODE_RISING_EDGE	241
10.33.6.4 EXT_CH_MODE_FALLING_EDGE	241
10.33.6.5 EXT_CH_MODE_BOTH_EDGES	241
10.33.6.6 EXT_CH_MODE_AUTOSTART	241
10.33.6.7 extChannelEnable	241
10.33.6.8 extChannelDisable	241
10.33.6.9 extSetChannelMode	241
10.33.6.10EXT_MAX_CHANNELS	242
10.33.6.11PLATFORM_EXT_USE_EXT1	242
10.33.7 Typedef Documentation	242
10.33.7.1 EXTDriver	242
10.33.7.2 expchannel_t	242
10.33.7.3 extcallback_t	242
10.33.8 Enumeration Type Documentation	242
10.33.8.1 extstate_t	242
10.34GPT Driver	243
10.34.1 Detailed Description	243

10.34.2 Driver State Machine	243
10.34.3 GPT Operations.	243
10.34.4 Function Documentation	245
10.34.4.1 gptInit	245
10.34.4.2 gptObjectInit	246
10.34.4.3 gptStart	246
10.34.4.4 gptStop	246
10.34.4.5 gptStartContinuous	247
10.34.4.6 gptStartContinuousl	247
10.34.4.7 gptChangeInterval	248
10.34.4.8 gptStartOneShot	248
10.34.4.9 gptStartOneShotl	249
10.34.4.10gptStopTimer	249
10.34.4.11gptStopTimerl	250
10.34.4.12gptPolledDelay	250
10.34.4.13gpt_lld_init	251
10.34.4.14gpt_lld_start	251
10.34.4.15gpt_lld_stop	251
10.34.4.16gpt_lld_start_timer	252
10.34.4.17gpt_lld_stop_timer	252
10.34.4.18gpt_lld_polled_delay	252
10.34.5 Variable Documentation	252
10.34.5.1 GPTD1	252
10.34.6 Define Documentation	252
10.34.6.1 gptChangeInterval	252
10.34.6.2 STM32_GPT_USE_TIM1	253
10.34.6.3 gpt_lld_change_interval	253
10.34.7 Typedef Documentation	254
10.34.7.1 GPTDriver	254
10.34.7.2 gptcallback_t	254
10.34.7.3 gptfreq_t	254
10.34.7.4 gptcnt_t	254
10.34.8 Enumeration Type Documentation	254
10.34.8.1 gptstate_t	254
10.35 HAL Driver	254
10.35.1 Detailed Description	254
10.35.2 Function Documentation	256
10.35.2.1 hallInit	256
10.35.2.2 hallsCounterWithin	257
10.35.2.3 halPolledDelay	258

10.35.2.4 hal_lld_init	259
10.35.2.5 platform_early_init	259
10.35.3 Define Documentation	259
10.35.3.1 S2RTT	259
10.35.3.2 MS2RTT	260
10.35.3.3 US2RTT	260
10.35.3.4 RTT2S	260
10.35.3.5 RTT2MS	261
10.35.3.6 RTT2US	261
10.35.3.7 halGetCounterValue	261
10.35.3.8 halGetCounterFrequency	261
10.35.3.9 HAL_IMPLEMENTED_COUNTERS	262
10.35.3.10 hal_lld_get_counter_value	262
10.35.3.11 hal_lld_get_counter_frequency	262
10.35.4 Typedef Documentation	262
10.35.4.1 halclock_t	262
10.35.4.2 halrtcnt_t	262
10.36 I2C Driver	263
10.36.1 Detailed Description	263
10.36.2 Driver State Machine	263
10.36.3 Function Documentation	265
10.36.3.1 i2cInit	265
10.36.3.2 i2cObjectInit	266
10.36.3.3 i2cStart	266
10.36.3.4 i2cStop	267
10.36.3.5 i2cGetErrors	267
10.36.3.6 i2cMasterTransmitTimeout	267
10.36.3.7 i2cMasterReceiveTimeout	268
10.36.3.8 i2cAcquireBus	269
10.36.3.9 i2cReleaseBus	270
10.36.3.10 i2c_lld_init	270
10.36.3.11 i2c_lld_start	270
10.36.3.12 i2c_lld_stop	271
10.36.3.13 i2c_lld_master_receive_timeout	271
10.36.3.14 i2c_lld_master_transmit_timeout	271
10.36.4 Variable Documentation	272
10.36.4.1 I2CD1	272
10.36.5 Define Documentation	272
10.36.5.1 I2CD_NO_ERROR	272
10.36.5.2 I2CD_BUS_ERROR	272

10.36.5.3 I2CD_ARBITRATION_LOST	272
10.36.5.4 I2CD_ACK_FAILURE	272
10.36.5.5 I2CD_OVERRUN	272
10.36.5.6 I2CD_PEC_ERROR	273
10.36.5.7 I2CD_TIMEOUT	273
10.36.5.8 I2CD_SMB_ALERT	273
10.36.5.9 I2C_USE_MUTUAL_EXCLUSION	273
10.36.5.10 I2cMasterTransmit	273
10.36.5.11 I2cMasterReceive	273
10.36.5.12 PLATFORM_I2C_USE_I2C1	273
10.36.5.13 I2c_lld_get_errors	273
10.36.6 Typedef Documentation	274
10.36.6.1 i2caddr_t	274
10.36.6.2 i2cflags_t	274
10.36.6.3 I2CDriver	274
10.36.7 Enumeration Type Documentation	274
10.36.7.1 i2cstate_t	274
10.37 I2S Driver	274
10.37.1 Driver State Machine	274
10.38 ICU Driver	274
10.38.1 Detailed Description	274
10.38.2 Driver State Machine	275
10.38.3 ICU Operations	275
10.38.4 Function Documentation	277
10.38.4.1 iculinit	277
10.38.4.2 icuObjectInit	278
10.38.4.3 icuStart	278
10.38.4.4 icuStop	278
10.38.4.5 icuEnable	279
10.38.4.6 icuDisable	279
10.38.4.7 icu_lld_init	280
10.38.4.8 icu_lld_start	280
10.38.4.9 icu_lld_stop	280
10.38.4.10 cu_lld_enable	281
10.38.4.11 cu_lld_disable	281
10.38.4.12 cu_lld_get_width	281
10.38.4.13 cu_lld_get_period	281
10.38.5 Variable Documentation	282
10.38.5.1 ICUD1	282
10.38.6 Define Documentation	282

10.38.6.1 icuEnable	282
10.38.6.2 icuDisable	282
10.38.6.3 icuGetWidth	282
10.38.6.4 icuGetPeriod	282
10.38.6.5 _icu_isr_invoke_width_cb	283
10.38.6.6 _icu_isr_invoke_period_cb	283
10.38.6.7 _icu_isr_invoke_overflow_cb	284
10.38.6.8 PLATFORM_ICU_USE_ICU1	284
10.38.7 Typedef Documentation	284
10.38.7.1 ICUDriver	284
10.38.7.2 icucallback_t	284
10.38.7.3 icufreq_t	284
10.38.7.4 icucnt_t	284
10.38.8 Enumeration Type Documentation	284
10.38.8.1 icustate_t	284
10.38.8.2 icumode_t	285
10.39 Abstract I/O Block Device	285
10.39.1 Detailed Description	285
10.39.2 Driver State Machine	285
10.39.3 Define Documentation	286
10.39.3.1 _base_block_device_methods	286
10.39.3.2 _base_block_device_data	287
10.39.3.3 blkGetDriverState	287
10.39.3.4 blkIsTransferring	287
10.39.3.5 blkIsInserted	288
10.39.3.6 blkIsWriteProtected	288
10.39.3.7 blkConnect	289
10.39.3.8 blkDisconnect	289
10.39.3.9 blkRead	289
10.39.3.10blkWrite	290
10.39.3.11blkSync	290
10.39.3.12blkGetInfo	290
10.39.4 Enumeration Type Documentation	291
10.39.4.1 blkstate_t	291
10.40 Abstract I/O Channel	291
10.40.1 Detailed Description	291
10.40.2 Define Documentation	293
10.40.2.1 _base_channel_methods	293
10.40.2.2 _base_channel_data	293
10.40.2.3 chnPutTimeout	293

10.40.2.4 chnGetTimeout	294
10.40.2.5 chnWrite	294
10.40.2.6 chnWriteTimeout	294
10.40.2.7 chnRead	295
10.40.2.8 chnReadTimeout	295
10.40.2.9 CHN_NO_ERROR	296
10.40.2.10 CHN_CONNECTED	296
10.40.2.11 CHN_DISCONNECTED	296
10.40.2.12 CHN_INPUT_AVAILABLE	296
10.40.2.13 CHN_OUTPUT_EMPTY	296
10.40.2.14 CHN_TRANSMISSION_END	296
10.40.2.15 base_asynchronous_channel_methods	296
10.40.2.16 base_asynchronous_channel_data	296
10.40.2.17 chnGetEventSource	296
10.40.2.18 chnAddFlags!	297
10.41 MAC Driver	297
10.41.1 Detailed Description	297
10.41.2 Function Documentation	299
10.41.2.1 macInit	299
10.41.2.2 macObjectInit	300
10.41.2.3 macStart	300
10.41.2.4 macStop	301
10.41.2.5 macWaitTransmitDescriptor	301
10.41.2.6 macReleaseTransmitDescriptor	302
10.41.2.7 macWaitReceiveDescriptor	302
10.41.2.8 macReleaseReceiveDescriptor	303
10.41.2.9 macPollLinkStatus	303
10.41.2.10 mac_lld_init	304
10.41.2.11 mac_lld_start	304
10.41.2.12 mac_lld_stop	305
10.41.2.13 mac_lld_get_transmit_descriptor	305
10.41.2.14 mac_lld_release_transmit_descriptor	305
10.41.2.15 mac_lld_get_receive_descriptor	305
10.41.2.16 mac_lld_release_receive_descriptor	306
10.41.2.17 mac_lld_poll_link_status	306
10.41.2.18 mac_lld_write_transmit_descriptor	306
10.41.2.19 mac_lld_read_receive_descriptor	307
10.41.2.20 mac_lld_get_next_transmit_buffer	307
10.41.2.21 mac_lld_get_next_receive_buffer	307
10.41.3 Variable Documentation	308

10.41.3.1 ETHD1	308
10.41.4 Define Documentation	308
10.41.4.1 MAC_USE_ZERO_COPY	308
10.41.4.2 MAC_USE_EVENTS	308
10.41.4.3 macGetReceiveEventSource	308
10.41.4.4 macWriteTransmitDescriptor	308
10.41.4.5 macReadReceiveDescriptor	309
10.41.4.6 macGetNextTransmitBuffer	309
10.41.4.7 macGetNextReceiveBuffer	309
10.41.4.8 MAC_SUPPORTS_ZERO_COPY	310
10.41.4.9 PLATFORM_MAC_USE_MAC1	310
10.41.5 Typedef Documentation	310
10.41.5.1 MACDriver	310
10.41.6 Enumeration Type Documentation	310
10.41.6.1 macstate_t	310
10.42 MMC over SPI Driver	310
10.42.1 Detailed Description	310
10.42.2 Driver State Machine	311
10.42.3 Driver Operations	311
10.42.4 Function Documentation	312
10.42.4.1 mmcInit	312
10.42.4.2 mmcObjectInit	312
10.42.4.3 mmcStart	312
10.42.4.4 mmcStop	313
10.42.4.5 mmcConnect	313
10.42.4.6 mmcDisconnect	314
10.42.4.7 mmcStartSequentialRead	315
10.42.4.8 mmcSequentialRead	316
10.42.4.9 mmcStopSequentialRead	316
10.42.4.10 mmcStartSequentialWrite	317
10.42.4.11 mmcSequentialWrite	318
10.42.4.12 mmcStopSequentialWrite	319
10.42.4.13 mmcSync	320
10.42.4.14 mmcGetInfo	320
10.42.4.15 mmcErase	320
10.42.5 Define Documentation	321
10.42.5.1 MMC_NICE_WAITING	321
10.42.5.2 _mmc_driver_methods	321
10.42.5.3 mmclsCardInserted	321
10.42.5.4 mmclsWriteProtected	322

10.43MMC/SD Block Device	322
10.43.1 Detailed Description	322
10.43.2 Function Documentation	325
10.43.2.1 mmcisdGetCapacity	325
10.43.3 Define Documentation	325
10.43.3.1 MMCSD_BLOCK_SIZE	325
10.43.3.2 MMCSD_R1_ERROR_MASK	325
10.43.3.3 MMCSD_CMD8_PATTERN	325
10.43.3.4 MMCSD_CSD_20_CRC_SLICE	326
10.43.3.5 _mmcisd_block_device_methods	326
10.43.3.6 _mmcisd_block_device_data	326
10.43.3.7 MMCSD_R1_ERROR	326
10.43.3.8 MMCSD_R1_STS	326
10.43.3.9 MMCSD_R1_IS_CARD_LOCKED	326
10.43.3.10 mmcisdGetCardCapacity	326
10.44PAL Driver	327
10.44.1 Detailed Description	327
10.44.2 Implementation Rules	327
10.44.2.1 Writing on input pads	327
10.44.2.2 Reading from output pads	327
10.44.2.3 Writing unused or unimplemented port bits	328
10.44.2.4 Reading from unused or unimplemented port bits	328
10.44.2.5 Reading or writing on pins associated to other functionalities	328
10.44.3 Function Documentation	331
10.44.3.1 palReadBus	331
10.44.3.2 palWriteBus	331
10.44.3.3 palSetBusMode	331
10.44.3.4 _pal_lld_init	332
10.44.3.5 _pal_lld_setgroupmode	332
10.44.4 Define Documentation	332
10.44.4.1 PAL_MODE_RESET	332
10.44.4.2 PAL_MODE_UNCONNECTED	332
10.44.4.3 PAL_MODE_INPUT	332
10.44.4.4 PAL_MODE_INPUT_PULLUP	333
10.44.4.5 PAL_MODE_INPUT_PULLDOWN	333
10.44.4.6 PAL_MODE_INPUT_ANALOG	333
10.44.4.7 PAL_MODE_OUTPUT_PUSH_PULL	333
10.44.4.8 PAL_MODE_OUTPUT_OPENDRAIN	333
10.44.4.9 PAL_LOW	333
10.44.4.10 PAL_HIGH	333

10.44.4.11PAL_PORT_BIT	333
10.44.4.12PAL_GROUP_MASK	333
10.44.4.13_IOBUS_DATA	334
10.44.4.14IOBUS_DECL	334
10.44.4.15palInit	334
10.44.4.16palReadPort	334
10.44.4.17palReadLatch	335
10.44.4.18palWritePort	335
10.44.4.19palSetPort	335
10.44.4.20palClearPort	336
10.44.4.21palTogglePort	336
10.44.4.22palReadGroup	336
10.44.4.23palWriteGroup	337
10.44.4.24palSetGroupMode	337
10.44.4.25palReadPad	337
10.44.4.26palWritePad	338
10.44.4.27palSetPad	338
10.44.4.28palClearPad	339
10.44.4.29palTogglePad	339
10.44.4.30palSetPadMode	340
10.44.4.31PAL_IOPORTS_WIDTH	340
10.44.4.32PAL_WHOLE_PORT	340
10.44.4.33IOPORT1	340
10.44.4.34pal_lld_init	340
10.44.4.35pal_lld_readport	340
10.44.4.36pal_lld_readlatch	341
10.44.4.37pal_lld_writeport	341
10.44.4.38pal_lld_setport	341
10.44.4.39pal_lld_clearport	342
10.44.4.40pal_lld_toggleport	342
10.44.4.41pal_lld_readgroup	342
10.44.4.42pal_lld_writegroup	343
10.44.4.43pal_lld_setgroupmode	343
10.44.4.44pal_lld_readpad	343
10.44.4.45pal_lld_writepad	344
10.44.4.46pal_lld_setpad	344
10.44.4.47pal_lld_clearpad	344
10.44.4.48pal_lld_togglepad	345
10.44.4.49pal_lld_setpadmode	345
10.44.5 Typedef Documentation	345

10.44.5.1 ioportmask_t	345
10.44.5.2 iomode_t	345
10.44.5.3 ioportid_t	345
10.45 PWM Driver	346
10.45.1 Detailed Description	346
10.45.2 Driver State Machine	346
10.45.3 PWM Operations	346
10.45.4 Function Documentation	349
10.45.4.1 pwmlInit	349
10.45.4.2 pwmObjectInit	349
10.45.4.3 pwmStart	349
10.45.4.4 pwmStop	350
10.45.4.5 pwmChangePeriod	350
10.45.4.6 pwmEnableChannel	351
10.45.4.7 pwmDisableChannel	351
10.45.4.8 pwm_lld_init	352
10.45.4.9 pwm_lld_start	352
10.45.4.10 pwm_lld_stop	352
10.45.4.11 pwm_lld_change_period	353
10.45.4.12 pwm_lld_enable_channel	353
10.45.4.13 pwm_lld_disable_channel	354
10.45.5 Variable Documentation	354
10.45.5.1 PWMD1	354
10.45.6 Define Documentation	354
10.45.6.1 PWM_OUTPUT_MASK	354
10.45.6.2 PWM_OUTPUT_DISABLED	354
10.45.6.3 PWM_OUTPUT_ACTIVE_HIGH	354
10.45.6.4 PWM_OUTPUT_ACTIVE_LOW	354
10.45.6.5 PWM_FRACTION_TO_WIDTH	355
10.45.6.6 PWM_DEGREES_TO_WIDTH	355
10.45.6.7 PWM_PERCENTAGE_TO_WIDTH	355
10.45.6.8 pwmChangePeriodl	356
10.45.6.9 pwmEnableChannell	356
10.45.6.10 pwmDisableChannell	357
10.45.6.11 pwmlsChannelEnabledl	357
10.45.6.12 PWM_CHANNELS	358
10.45.6.13 PLATFORM_PWM_USE_PWM1	358
10.45.6.14 pwm_lld_is_channel_enabled	358
10.45.7 Typedef Documentation	358
10.45.7.1 PWMDriver	358

10.45.7.2 <code>pwmcallback_t</code>	358
10.45.7.3 <code>pwmmode_t</code>	358
10.45.7.4 <code>pwmchannel_t</code>	358
10.45.7.5 <code>pwmcnt_t</code>	358
10.45.8 Enumeration Type Documentation	359
10.45.8.1 <code>pwmstate_t</code>	359
10.46 RTC Driver	359
10.46.1 Detailed Description	359
10.46.2 Function Documentation	360
10.46.2.1 <code>rtcInit</code>	360
10.46.2.2 <code>rtcSetTime</code>	360
10.46.2.3 <code>rtcGetTime</code>	360
10.46.2.4 <code>rtcGetTimeFat</code>	361
10.46.2.5 <code>rtcSetAlarm</code>	361
10.46.2.6 <code>rtcGetAlarm</code>	361
10.46.2.7 <code>rtcSetCallback</code>	361
10.46.3 Define Documentation	362
10.46.3.1 <code>rtcSetTimel</code>	362
10.46.3.2 <code>rtcGetTimel</code>	362
10.46.3.3 <code>rtcSetAlarml</code>	362
10.46.3.4 <code>rtcGetAlarml</code>	362
10.46.3.5 <code>rtcSetCallbackl</code>	363
10.46.4 Typedef Documentation	363
10.46.4.1 <code>RTCDriver</code>	363
10.46.4.2 <code>RTCTime</code>	363
10.47 SDC Driver	363
10.47.1 Detailed Description	363
10.47.2 Driver State Machine	363
10.47.3 Driver Operations	364
10.47.4 Function Documentation	367
10.47.4.1 <code>sdcInit</code>	367
10.47.4.2 <code>sdcObjectInit</code>	367
10.47.4.3 <code>sdcStart</code>	367
10.47.4.4 <code>sdcStop</code>	368
10.47.4.5 <code>sdcConnect</code>	368
10.47.4.6 <code>sdcDisconnect</code>	369
10.47.4.7 <code>sdcRead</code>	370
10.47.4.8 <code>sdcWrite</code>	371
10.47.4.9 <code>sdcGetAndClearErrors</code>	371
10.47.4.10 <code>sdcSync</code>	371

10.47.4.11sdcGetInfo	372
10.47.4.12sdcErase	372
10.47.4.13_sdc_wait_for_transfer_state	373
10.47.4.14sdc_lld_init	374
10.47.4.15sdc_lld_start	374
10.47.4.16sdc_lld_stop	374
10.47.4.17sdc_lld_start_clk	374
10.47.4.18sdc_lld_set_data_clk	375
10.47.4.19sdc_lld_stop_clk	375
10.47.4.20sdc_lld_set_bus_mode	375
10.47.4.21sdc_lld_send_cmd_none	375
10.47.4.22sdc_lld_send_cmd_short	375
10.47.4.23sdc_lld_send_cmd_short_crc	376
10.47.4.24sdc_lld_send_cmd_long_crc	376
10.47.4.25sdc_lld_read	377
10.47.4.26sdc_lld_write	377
10.47.4.27sdc_lld_sync	377
10.47.5 Variable Documentation	378
10.47.5.1 SDCD1	378
10.47.6 Define Documentation	378
10.47.6.1 SDC_MODE_CARDTYPE_MASK	378
10.47.6.2 SDC_MODE_CARDTYPE_SDV11	378
10.47.6.3 SDC_MODE_CARDTYPE_SDV20	378
10.47.6.4 SDC_MODE_CARDTYPE_MMC	378
10.47.6.5 SDC_MODE_HIGH_CAPACITY	378
10.47.6.6 SDC_NO_ERROR	378
10.47.6.7 SDC_CMD_CRC_ERROR	378
10.47.6.8 SDC_DATA_CRC_ERROR	378
10.47.6.9 SDC_DATA_TIMEOUT	379
10.47.6.10 SDC_COMMAND_TIMEOUT	379
10.47.6.11 SDC_TX_UNDERRUN	379
10.47.6.12 SDC_RX_OVERRUN	379
10.47.6.13 SDC_STARTBIT_ERROR	379
10.47.6.14 SDC_OVERFLOW_ERROR	379
10.47.6.15 SDC_INIT_RETRY	379
10.47.6.16 SDC_MMC_SUPPORT	379
10.47.6.17 SDC_NICE_WAITING	379
10.47.6.18 sdclsCardInserted	380
10.47.6.19 sdclsWriteProtected	380
10.47.6.20 PLATFORM_SDC_USE_SDC1	380

10.47.6.21_sdc_driver_methods	380
10.47.6.22MMCSD_R1_ERROR	381
10.47.6.23MMCSD_R1_STS	381
10.47.6.24MMCSD_R1_IS_CARD_LOCKED	381
10.47.7 Typedef Documentation	381
10.47.7.1 sdcmode_t	381
10.47.7.2 sdcflags_t	381
10.47.7.3 SDCDriver	381
10.47.8 Enumeration Type Documentation	381
10.47.8.1 sdcbusmode_t	381
10.48Serial Driver	381
10.48.1 Detailed Description	381
10.48.2 Driver State Machine	382
10.48.3 Function Documentation	384
10.48.3.1 sdInit	384
10.48.3.2 sdObjectInit	385
10.48.3.3 sdStart	385
10.48.3.4 sdStop	386
10.48.3.5 sdIncomingData	386
10.48.3.6 sdRequestData	387
10.48.3.7 sd_lld_init	388
10.48.3.8 sd_lld_start	388
10.48.3.9 sd_lld_stop	388
10.48.4 Variable Documentation	389
10.48.4.1 SD1	389
10.48.5 Define Documentation	389
10.48.5.1 SD_PARITY_ERROR	389
10.48.5.2 SD_FRAMING_ERROR	389
10.48.5.3 SD_OVERRUN_ERROR	389
10.48.5.4 SD_NOISE_ERROR	389
10.48.5.5 SD_BREAK_DETECTED	389
10.48.5.6 SERIAL_DEFAULT_BITRATE	389
10.48.5.7 SERIAL_BUFFERS_SIZE	389
10.48.5.8 _serial_driver_methods	390
10.48.5.9 sdPutWouldBlock	390
10.48.5.10sdGetWouldBlock	390
10.48.5.11sdPut	390
10.48.5.12sdPutTimeout	391
10.48.5.13sdGet	391
10.48.5.14sdGetTimeout	391

10.48.5.15sdWrite	391
10.48.5.16sdWriteTimeout	392
10.48.5.17sdAsynchronousWrite	392
10.48.5.18sdRead	392
10.48.5.19sdReadTimeout	393
10.48.5.20sdAsynchronousRead	393
10.48.5.21PLATFORM_SERIAL_USE_SD1	393
10.48.5.22_serial_driver_data	393
10.48.6 Ttypedef Documentation	394
10.48.6.1 SerialDriver	394
10.48.7 Eenumeration Type Documentation	394
10.48.7.1 sdstate_t	394
10.49Sserial over USB Driver	394
10.49.1 Ddetailed Description	394
10.49.2 Ddriver State Machine	394
10.49.3 Ffunction Documentation	396
10.49.3.1 sduInit	396
10.49.3.2 sduObjectInit	396
10.49.3.3 sduStart	397
10.49.3.4 sduStop	397
10.49.3.5 sduConfigureHookl	398
10.49.3.6 sduRequestsHook	399
10.49.3.7 sduDataTransmitted	399
10.49.3.8 sduDataReceived	399
10.49.3.9 sduInterruptTransmitted	400
10.49.4 Ddefine Documentation	400
10.49.4.1 SERIAL_USB_BUFFERS_SIZE	400
10.49.4.2 _serial_usb_driver_data	400
10.49.4.3 _serial_usb_driver_methods	401
10.49.5 Ttypedef Documentation	401
10.49.5.1 SerialUSBDriver	401
10.49.6 Eenumeration Type Documentation	401
10.49.6.1 sdustate_t	401
10.50SPI Driver	401
10.50.1 Ddetailed Description	401
10.50.2 Ddriver State Machine	401
10.50.3 Ffunction Documentation	405
10.50.3.1 spilinit	405
10.50.3.2 spiObjectInit	405
10.50.3.3 spiStart	405

10.50.3.4 spiStop	406
10.50.3.5 spiSelect	406
10.50.3.6 spiUnselect	407
10.50.3.7 spiStartIgnore	407
10.50.3.8 spiStartExchange	407
10.50.3.9 spiStartSend	408
10.50.3.10spiStartReceive	408
10.50.3.11spilgnore	409
10.50.3.12spiExchange	409
10.50.3.13spiSend	409
10.50.3.14spiReceive	410
10.50.3.15spiAcquireBus	410
10.50.3.16spiReleaseBus	411
10.50.3.17spi_lld_init	411
10.50.3.18spi_lld_start	412
10.50.3.19spi_lld_stop	412
10.50.3.20spi_lld_select	412
10.50.3.21spi_lld_unselect	412
10.50.3.22spi_lld_ignore	413
10.50.3.23spi_lld_exchange	413
10.50.3.24spi_lld_send	413
10.50.3.25spi_lld_receive	414
10.50.3.26spi_lld_polled_exchange	414
10.50.4 Variable Documentation	415
10.50.4.1 SPID1	415
10.50.5 Define Documentation	415
10.50.5.1 SPI_USE_WAIT	415
10.50.5.2 SPI_USE_MUTUAL_EXCLUSION	415
10.50.5.3 spiSelectl	415
10.50.5.4 spiUnselectl	415
10.50.5.5 spiStartIgnorel	416
10.50.5.6 spiStartExchangel	416
10.50.5.7 spiStartSendl	417
10.50.5.8 spiStartReceive l	417
10.50.5.9 spiPolledExchange	418
10.50.5.10_spi_wait_s	418
10.50.5.11_spi_wakeup_isr	419
10.50.5.12_spi_isr_code	419
10.50.5.13PLATFORM_SPI_USE_SPI1	420
10.50.6 Typedef Documentation	420

10.50.6.1 SPIDriver	420
10.50.6.2 spicallback_t	420
10.50.7 Enumeration Type Documentation	420
10.50.7.1 spistate_t	420
10.51 Time Measurement Driver	420
10.51.1 Detailed Description	420
10.51.2 Function Documentation	421
10.51.2.1 tmInit	421
10.51.2.2 tmObjectInit	421
10.51.3 Define Documentation	422
10.51.3.1 tmStartMeasurement	422
10.51.3.2 tmStopMeasurement	422
10.51.4 Typedef Documentation	422
10.51.4.1 TimeMeasurement	422
10.52 UART Driver	422
10.52.1 Detailed Description	423
10.52.2 Driver State Machine	423
10.52.2.1 Transmitter sub State Machine	423
10.52.2.2 Receiver sub State Machine	424
10.52.3 Function Documentation	426
10.52.3.1 uartInit	426
10.52.3.2 uartObjectInit	427
10.52.3.3 uartStart	427
10.52.3.4 uartStop	427
10.52.3.5 uartStartSend	428
10.52.3.6 uartStartSendl	428
10.52.3.7 uartStopSend	429
10.52.3.8 uartStopSendl	430
10.52.3.9 uartStartReceive	430
10.52.3.10uartStartReceive1	431
10.52.3.11uartStopReceive	432
10.52.3.12uartStopReceive1	432
10.52.3.13uart_lld_init	433
10.52.3.14uart_lld_start	433
10.52.3.15uart_lld_stop	434
10.52.3.16uart_lld_start_send	434
10.52.3.17uart_lld_stop_send	434
10.52.3.18uart_lld_start_receive	434
10.52.3.19uart_lld_stop_receive	435
10.52.4 Variable Documentation	435

10.52.4.1 UARTD1	435
10.52.5 Define Documentation	435
10.52.5.1 UART_NO_ERROR	435
10.52.5.2 UART_PARITY_ERROR	435
10.52.5.3 UART_FRAMING_ERROR	435
10.52.5.4 UART_OVERRUN_ERROR	436
10.52.5.5 UART_NOISE_ERROR	436
10.52.5.6 UART_BREAK_DETECTED	436
10.52.5.7 PLATFORM_UART_USE_UART1	436
10.52.6 Typedef Documentation	436
10.52.6.1 uartflags_t	436
10.52.6.2 UARTDriver	436
10.52.6.3 uartcb_t	436
10.52.6.4 uartccb_t	436
10.52.6.5 uarteccb_t	436
10.52.7 Enumeration Type Documentation	437
10.52.7.1 uartstate_t	437
10.52.7.2 uarttxstate_t	437
10.52.7.3 uartrxstate_t	437
10.53 USB Driver	437
10.53.1 Detailed Description	437
10.53.2 Driver State Machine	437
10.53.3 USB Operations	438
10.53.3.1 USB Implementation	438
10.53.3.2 USB Endpoints	438
10.53.3.3 USB Callbacks	439
10.53.4 Function Documentation	444
10.53.4.1 usblInit	444
10.53.4.2 usbObjectInit	444
10.53.4.3 usbStart	445
10.53.4.4 usbStop	445
10.53.4.5 usblInitEndpoint1	445
10.53.4.6 usbDisableEndpoints1	446
10.53.4.7 usbPrepareReceive	447
10.53.4.8 usbPrepareTransmit	447
10.53.4.9 usbPrepareQueuedReceive	448
10.53.4.10usbPrepareQueuedTransmit	449
10.53.4.11usbStartReceive1	449
10.53.4.12usbStartTransmit1	450
10.53.4.13usbStallReceive1	451

10.53.4.14usbStallTransmitl	451
10.53.4.15_usb_reset	452
10.53.4.16_usb_ep0setup	453
10.53.4.17_usb_ep0in	453
10.53.4.18_usb_ep0out	454
10.53.4.19usb_lld_init	455
10.53.4.20usb_lld_start	455
10.53.4.21usb_lld_stop	456
10.53.4.22usb_lld_reset	456
10.53.4.23usb_lld_set_address	456
10.53.4.24usb_lld_init_endpoint	456
10.53.4.25usb_lld_disable_endpoints	457
10.53.4.26usb_lld_get_status_out	457
10.53.4.27usb_lld_get_status_in	457
10.53.4.28usb_lld_read_setup	458
10.53.4.29usb_lld_prepare_receive	458
10.53.4.30usb_lld_prepare_transmit	458
10.53.4.31usb_lld_start_out	458
10.53.4.32usb_lld_start_in	459
10.53.4.33usb_lld_stall_out	459
10.53.4.34usb_lld_stall_in	459
10.53.4.35usb_lld_clear_out	459
10.53.4.36usb_lld_clear_in	460
10.53.5 Variable Documentation	460
10.53.5.1 USBD1	460
10.53.5.2 in	460
10.53.5.3 out	460
10.53.6 Define Documentation	460
10.53.6.1 USB_DESC_INDEX	460
10.53.6.2 USB_DESC_BYTE	460
10.53.6.3 USB_DESC_WORD	460
10.53.6.4 USB_DESC_BCD	460
10.53.6.5 USB_DESC_DEVICE	461
10.53.6.6 USB_DESC_CONFIGURATION	461
10.53.6.7 USB_DESC_INTERFACE	461
10.53.6.8 USB_DESC_INTERFACE_ASSOCIATION	461
10.53.6.9 USB_DESC_ENDPOINT	462
10.53.6.10USB_EP_MODE_TYPE	462
10.53.6.11USB_EP_MODE_TYPE_CTRL	462
10.53.6.12USB_EP_MODE_TYPE_ISOC	462

10.53.6.13USB_EP_MODE_TYPE_BULK	462
10.53.6.14USB_EP_MODE_TYPE_INTR	462
10.53.6.15USB_EP_MODE_LINEAR_BUFFER	462
10.53.6.16USB_EP_MODE_QUEUE_BUFFER	462
10.53.6.17usbGetDriverStatel	463
10.53.6.18usbFetchWord	463
10.53.6.19usbConnectBus	463
10.53.6.20usbDisconnectBus	463
10.53.6.21usbGetFrameNumber	463
10.53.6.22usbGetTransmitStatusl	464
10.53.6.23usbGetReceiveStatusl	464
10.53.6.24usbGetReceiveTransactionSizel	464
10.53.6.25usbSetupTransfer	465
10.53.6.26usbReadSetup	465
10.53.6.27_usb_isr_invoke_event_cb	466
10.53.6.28_usb_isr_invoke_sof_cb	466
10.53.6.29_usb_isr_invoke_setup_cb	466
10.53.6.30_usb_isr_invoke_in_cb	467
10.53.6.31_usb_isr_invoke_out_cb	467
10.53.6.32USB_MAX_ENDPOINTS	467
10.53.6.33USB_EP0_STATUS_STAGE	467
10.53.6.34USB_SET_ADDRESS_MODE	467
10.53.6.35USB_SET_ADDRESS_ACK_HANDLING	468
10.53.6.36PLATFORM_USB_USE_USB1	468
10.53.6.37usb_lld_get_transaction_size	468
10.53.6.38usb_lld_connect_bus	468
10.53.6.39usb_lld_disconnect_bus	468
10.53.7 Typedef Documentation	468
10.53.7.1 USBDriver	468
10.53.7.2 usbep_t	469
10.53.7.3 usbcallback_t	469
10.53.7.4 usbepcallback_t	469
10.53.7.5 usbeventcb_t	469
10.53.7.6 usbreqhandler_t	469
10.53.7.7 usbgetdescriptor_t	469
10.53.8 Enumeration Type Documentation	470
10.53.8.1 usbstate_t	470
10.53.8.2 usbepstatus_t	470
10.53.8.3 usbep0state_t	470
10.53.8.4 usbevent_t	470

10.54 HAL	471
10.54.1 Detailed Description	471
10.54.2 HAL Device Drivers Architecture	471
10.54.2.1 Diagram	472
10.55 Configuration	473
10.55.1 Detailed Description	473
10.55.2 Define Documentation	475
10.55.2.1 HAL_USE_TM	475
10.55.2.2 HAL_USE_PAL	475
10.55.2.3 HAL_USE_ADC	475
10.55.2.4 HAL_USE_CAN	475
10.55.2.5 HAL_USE_EXT	475
10.55.2.6 HAL_USE_GPT	476
10.55.2.7 HAL_USE_I2C	476
10.55.2.8 HAL_USE_ICU	476
10.55.2.9 HAL_USE_MAC	476
10.55.2.10 HAL_USE_MMC_SPI	476
10.55.2.11 HAL_USE_PWM	476
10.55.2.12 HAL_USE_RTC	476
10.55.2.13 HAL_USE_SDC	476
10.55.2.14 HAL_USE_SERIAL	476
10.55.2.15 HAL_USE_SERIAL_USB	476
10.55.2.16 HAL_USE_SPI	476
10.55.2.17 HAL_USE_UART	476
10.55.2.18 HAL_USE_USB	477
10.55.2.19 ADC_USE_WAIT	477
10.55.2.20 ADC_USE_MUTUAL_EXCLUSION	477
10.55.2.21 CAN_USE_SLEEP_MODE	477
10.55.2.22 I2C_USE_MUTUAL_EXCLUSION	477
10.55.2.23 MAC_USE_ZERO_COPY	477
10.55.2.24 MAC_USE_EVENTS	477
10.55.2.25 MMC_NICE_WAITING	477
10.55.2.26 SDC_INIT_RETRY	477
10.55.2.27 SDC_MMC_SUPPORT	478
10.55.2.28 SDC_NICE_WAITING	478
10.55.2.29 SERIAL_DEFAULT_BITRATE	478
10.55.2.30 SERIAL_BUFFERS_SIZE	478
10.55.2.31 SERIAL_USB_BUFFERS_SIZE	478
10.55.2.32 SPI_USE_WAIT	478
10.55.2.33 SPI_USE_MUTUAL_EXCLUSION	478

10.56 Various	479
10.56.1 Detailed Description	479
10.57 C++ Wrapper	479
10.57.1 Detailed Description	479
10.58 Memory Streams	479
10.58.1 Detailed Description	479
10.58.2 Function Documentation	480
10.58.2.1 msObjectInit	480
10.58.3 Define Documentation	480
10.58.3.1 _memory_stream_data	480
10.59 Periodic Events Timer	480
10.59.1 Detailed Description	480
10.59.2 Function Documentation	481
10.59.2.1 evtStart	481
10.59.2.2 evtStop	481
10.59.3 Define Documentation	481
10.59.3.1 evtInit	482
10.60 Command Shell	482
10.60.1 Detailed Description	482
10.60.2 Function Documentation	483
10.60.2.1 shellInit	483
10.60.2.2 shellExit	483
10.60.2.3 shellCreate	484
10.60.2.4 shellCreateStatic	484
10.60.2.5 shellGetLine	485
10.60.3 Variable Documentation	485
10.60.3.1 shell_terminated	485
10.60.4 Define Documentation	485
10.60.4.1 SHELL_MAX_LINE_LENGTH	485
10.60.4.2 SHELL_MAX_ARGUMENTS	486
10.60.5 Typedef Documentation	486
10.60.5.1 shellcmd_t	486
10.61 RTC time conversion utilities	486
10.61.1 Detailed Description	486
10.61.2 Function Documentation	486
10.61.2.1 rtcGetTimeTm	486
10.61.2.2 rtcSetTimeTm	487
10.61.2.3 rtcGetTimeUnixSec	487
10.61.2.4 rtcSetTimeUnixSec	488
10.61.2.5 rtcGetTimeUnixUsec	488

10.61.2.6 rtcGetTimeFatFromCounter	489
10.61.2.7 rtcGetTimeFat	489
10.62 System formatted print	490
10.62.1 Detailed Description	490
10.62.2 Function Documentation	490
10.62.2.1 chvprintf	490
10.62.2.2 chsnprintf	491
10.62.3 Define Documentation	491
10.62.3.1 CHPRINTF_USE_FLOAT	491
10.63 Test Runtime	492
10.63.1 Detailed Description	492
10.63.2 Function Documentation	493
10.63.2.1 test_printf	493
10.63.2.2 test_print	493
10.63.2.3 test_printfln	493
10.63.2.4 test_emit_token	493
10.63.2.5 test_terminate_threads	494
10.63.2.6 test_wait_threads	494
10.63.2.7 test_cpu_pulse	494
10.63.2.8 test_wait_tick	494
10.63.2.9 test_start_timer	495
10.63.2.10 TestThread	495
10.63.3 Variable Documentation	495
10.63.3.1 test_timer_done	496
10.63.4 Define Documentation	496
10.63.4.1 DELAY_BETWEEN_TESTS	496
10.63.4.2 TEST_NO_BENCHMARKS	496
10.63.4.3 test_fail	496
10.63.4.4 test_assert	496
10.63.4.5 test_assert_lock	496
10.63.4.6 test_assert_sequence	497
10.63.4.7 test_assert_time_window	497
10.64 External Components	497
11 Namespace Documentation	498
11.1 chibios_rt Namespace Reference	498
11.1.1 Detailed Description	498
12 Data Structure Documentation	500
12.1 ADCConfig Struct Reference	500
12.1.1 Detailed Description	500

12.2 ADCConversionGroup Struct Reference	500
12.2.1 Detailed Description	500
12.2.2 Field Documentation	502
12.2.2.1 circular	502
12.2.2.2 num_channels	502
12.2.2.3 end_cb	502
12.2.2.4 error_cb	502
12.3 ADCDriver Struct Reference	502
12.3.1 Detailed Description	502
12.3.2 Field Documentation	504
12.3.2.1 state	504
12.3.2.2 config	504
12.3.2.3 samples	504
12.3.2.4 depth	504
12.3.2.5 grpp	504
12.3.2.6 thread	504
12.3.2.7 mutex	504
12.4 BaseAsynchronousChannel Struct Reference	504
12.4.1 Detailed Description	504
12.4.2 Field Documentation	506
12.4.2.1 vmt	506
12.5 BaseAsynchronousChannelVMT Struct Reference	506
12.5.1 Detailed Description	506
12.6 BaseBlockDevice Struct Reference	508
12.6.1 Detailed Description	508
12.6.2 Field Documentation	510
12.6.2.1 vmt	510
12.7 BaseBlockDeviceVMT Struct Reference	510
12.7.1 Detailed Description	510
12.8 BaseChannel Struct Reference	510
12.8.1 Detailed Description	510
12.8.2 Field Documentation	512
12.8.2.1 vmt	512
12.9 BaseChannelVMT Struct Reference	512
12.9.1 Detailed Description	512
12.10BaseFileStream Struct Reference	514
12.10.1 Detailed Description	514
12.10.2 Field Documentation	515
12.10.2.1 vmt	515
12.11BaseFileStreamVMT Struct Reference	515

12.11.1 Detailed Description	515
12.12BaseSequentialStream Struct Reference	516
12.12.1 Detailed Description	516
12.12.2 Field Documentation	518
12.12.2.1 vmt	518
12.13chibios_rt::BaseSequentialStreamInterface Class Reference	518
12.13.1 Detailed Description	518
12.13.2 Member Function Documentation	518
12.13.2.1 write	518
12.13.2.2 read	519
12.13.2.3 put	519
12.13.2.4 get	519
12.14BaseSequentialStreamVMT Struct Reference	520
12.14.1 Detailed Description	520
12.15chibios_rt::BaseStaticThread< N > Class Template Reference	520
12.15.1 Detailed Description	520
12.15.2 Constructor & Destructor Documentation	523
12.15.2.1 BaseStaticThread	523
12.15.3 Member Function Documentation	523
12.15.3.1 start	523
12.16chibios_rt::BaseThread Class Reference	523
12.16.1 Detailed Description	523
12.16.2 Constructor & Destructor Documentation	527
12.16.2.1 BaseThread	527
12.16.3 Member Function Documentation	527
12.16.3.1 main	527
12.16.3.2 start	527
12.16.3.3 setName	527
12.16.3.4 setPriority	528
12.16.3.5 exit	528
12.16.3.6 exitS	529
12.16.3.7 shouldTerminate	529
12.16.3.8 sleep	530
12.16.3.9 sleepUntil	530
12.16.3.10yield	531
12.16.3.11waitMessage	531
12.16.3.12getAndClearEvents	532
12.16.3.13addEvents	532
12.16.3.14waitOneEvent	533
12.16.3.15waitAnyEvent	533

12.16.3.10	waitAllEvents	534
12.16.3.17	waitOneEventTimeout	534
12.16.3.18	waitAnyEventTimeout	535
12.16.3.19	waitAllEventsTimeout	536
12.16.3.20	dispatchEvents	536
12.16.3.21	unlockMutex	537
12.16.3.22	unlockMutexS	537
12.16.3.23	unlockAllMutexes	538
12.17	BinarySemaphore Struct Reference	538
12.17.1	Detailed Description	538
12.18	chibios_rt::BinarySemaphore Class Reference	540
12.18.1	Detailed Description	540
12.18.2	Constructor & Destructor Documentation	541
12.18.2.1	BinarySemaphore	541
12.18.3	Member Function Documentation	541
12.18.3.1	wait	541
12.18.3.2	waitForS	541
12.18.3.3	waitForTimeout	542
12.18.3.4	waitForTimeoutS	542
12.18.3.5	reset	542
12.18.3.6	resetI	543
12.18.3.7	signal	543
12.18.3.8	signalI	543
12.18.3.9	getStateI	543
12.18.4	Field Documentation	544
12.18.4.1	bsem	544
12.19	BlockDeviceInfo Struct Reference	544
12.19.1	Detailed Description	544
12.19.2	Field Documentation	544
12.19.2.1	blk_size	544
12.19.2.2	blk_num	544
12.20	CANConfig Struct Reference	544
12.20.1	Detailed Description	544
12.21	CANDriver Struct Reference	545
12.21.1	Detailed Description	545
12.21.2	Field Documentation	546
12.21.2.1	state	546
12.21.2.2	config	546
12.21.2.3	txsem	546
12.21.2.4	rxsem	546

12.21.2.5 rxfull_event	546
12.21.2.6 txempty_event	547
12.21.2.7 error_event	547
12.21.2.8 sleep_event	547
12.21.2.9 wakeup_event	547
12.22CANFilter Struct Reference	547
12.22.1 Detailed Description	547
12.23CANRxFrame Struct Reference	547
12.23.1 Detailed Description	547
12.23.2 Field Documentation	548
12.23.2.1 DLC	548
12.23.2.2 RTR	548
12.23.2.3 IDE	548
12.23.2.4 SID	548
12.23.2.5 EID	548
12.23.2.6 data8	548
12.23.2.7 data16	548
12.23.2.8 data32	548
12.24CANTxFrame Struct Reference	548
12.24.1 Detailed Description	548
12.24.2 Field Documentation	548
12.24.2.1 DLC	548
12.24.2.2 RTR	549
12.24.2.3 IDE	549
12.24.2.4 SID	549
12.24.2.5 EID	549
12.24.2.6 data8	549
12.24.2.7 data16	549
12.24.2.8 data32	549
12.25cdc_linecoding_t Struct Reference	549
12.25.1 Detailed Description	549
12.26ch_swc_event_t Struct Reference	549
12.26.1 Detailed Description	549
12.26.2 Field Documentation	551
12.26.2.1 se_time	551
12.26.2.2 se_tp	551
12.26.2.3 se_wtobjp	551
12.26.2.4 se_state	551
12.27ch_trace_buffer_t Struct Reference	551
12.27.1 Detailed Description	551

12.27.2 Field Documentation	553
12.27.2.1 tb_size	553
12.27.2.2 tb_ptr	553
12.27.2.3 tb_buffer	553
12.28chdebug_t Struct Reference	553
12.28.1 Detailed Description	553
12.28.2 Field Documentation	554
12.28.2.1 ch_identifier	554
12.28.2.2 ch_zero	554
12.28.2.3 ch_size	554
12.28.2.4 ch_version	554
12.28.2.5 ch_ptrsize	554
12.28.2.6 ch_timesize	554
12.28.2.7 ch_threadsize	554
12.28.2.8 cf_off_prio	554
12.28.2.9 cf_off_ctx	555
12.28.2.10cf_off_newer	555
12.28.2.11cf_off_older	555
12.28.2.12cf_off_name	555
12.28.2.13cf_off_stklimit	555
12.28.2.14cf_off_state	555
12.28.2.15cf_off_flags	555
12.28.2.16cf_off_refs	555
12.28.2.17cf_off_preempt	555
12.28.2.18cf_off_time	555
12.29CondVar Struct Reference	555
12.29.1 Detailed Description	555
12.29.2 Field Documentation	556
12.29.2.1 c_queue	557
12.30chibios_rt::CondVar Class Reference	557
12.30.1 Detailed Description	557
12.30.2 Constructor & Destructor Documentation	558
12.30.2.1 CondVar	558
12.30.3 Member Function Documentation	558
12.30.3.1 signal	558
12.30.3.2 signall	558
12.30.3.3 broadcast	559
12.30.3.4 broadcastl	559
12.30.3.5 wait	560
12.30.3.6 waitS	560

12.30.3.7 waitTimeout	561
12.30.4 Field Documentation	562
12.30.4.1 condvar	562
12.31 context Struct Reference	562
12.31.1 Detailed Description	562
12.32 chibios_rt::Core Class Reference	562
12.32.1 Detailed Description	562
12.32.2 Member Function Documentation	563
12.32.2.1 alloc	563
12.32.2.2 allocI	563
12.32.2.3 getStatus	564
12.33 chibios_rt::CounterSemaphore Class Reference	564
12.33.1 Detailed Description	564
12.33.2 Constructor & Destructor Documentation	566
12.33.2.1 CounterSemaphore	566
12.33.3 Member Function Documentation	566
12.33.3.1 reset	566
12.33.3.2 resetI	567
12.33.3.3 wait	568
12.33.3.4 waitS	568
12.33.3.5 waitTimeout	569
12.33.3.6 waitTimeoutS	569
12.33.3.7 signal	570
12.33.3.8 signall	570
12.33.3.9 addCounterI	571
12.33.3.10 getCounterI	571
12.33.3.11 signalWait	572
12.33.4 Field Documentation	572
12.33.4.1 sem	572
12.34 EventListener Struct Reference	572
12.34.1 Detailed Description	572
12.34.2 Field Documentation	574
12.34.2.1 el_next	574
12.34.2.2 el_listener	574
12.34.2.3 el_mask	574
12.34.2.4 el_flags	574
12.35 EventSource Struct Reference	574
12.35.1 Detailed Description	574
12.35.2 Field Documentation	576
12.35.2.1 es_next	576

12.36EvTimer Struct Reference	576
12.36.1 Detailed Description	576
12.37chibios_rt::EvtListener Class Reference	576
12.37.1 Detailed Description	577
12.37.2 Member Function Documentation	577
12.37.2.1 getAndClearFlags	577
12.37.2.2 getAndClearFlagsl	577
12.37.3 Field Documentation	578
12.37.3.1 ev_listener	578
12.38chibios_rt::EvtSource Class Reference	578
12.38.1 Detailed Description	578
12.38.2 Constructor & Destructor Documentation	578
12.38.2.1 EvtSource	578
12.38.3 Member Function Documentation	579
12.38.3.1 registerOne	579
12.38.3.2 registerMask	579
12.38.3.3 unregister	579
12.38.3.4 broadcastFlags	580
12.38.3.5 broadcastFlagsl	580
12.38.4 Field Documentation	581
12.38.4.1 ev_source	581
12.39EXTChannelConfig Struct Reference	581
12.39.1 Detailed Description	581
12.39.2 Field Documentation	582
12.39.2.1 mode	582
12.39.2.2 cb	582
12.40EXTConfig Struct Reference	582
12.40.1 Detailed Description	582
12.40.2 Field Documentation	583
12.40.2.1 channels	583
12.41extctx Struct Reference	583
12.41.1 Detailed Description	583
12.42EXTDriver Struct Reference	584
12.42.1 Detailed Description	584
12.42.2 Field Documentation	584
12.42.2.1 state	584
12.42.2.2 config	584
12.43GenericQueue Struct Reference	585
12.43.1 Detailed Description	585
12.43.2 Field Documentation	587

12.43.2.1 <code>q_waiting</code>	587
12.43.2.2 <code>q_counter</code>	587
12.43.2.3 <code>q_buffer</code>	587
12.43.2.4 <code>q_top</code>	587
12.43.2.5 <code>q_wptr</code>	587
12.43.2.6 <code>q_rdptr</code>	587
12.43.2.7 <code>q_notify</code>	587
12.43.2.8 <code>q_link</code>	587
12.44GPTConfig Struct Reference	588
12.44.1 Detailed Description	588
12.44.2 Field Documentation	588
12.44.2.1 <code>frequency</code>	588
12.44.2.2 <code>callback</code>	589
12.45GPTDriver Struct Reference	589
12.45.1 Detailed Description	589
12.45.2 Field Documentation	589
12.45.2.1 <code>state</code>	589
12.45.2.2 <code>config</code>	590
12.46heap_header Union Reference	590
12.46.1 Detailed Description	590
12.46.2 Field Documentation	590
12.46.2.1 <code>next</code>	590
12.46.2.2 <code>heap</code>	590
12.46.2.3 <code>u</code>	590
12.46.2.4 <code>size</code>	591
12.47I2CConfig Struct Reference	591
12.47.1 Detailed Description	591
12.48I2CDriver Struct Reference	591
12.48.1 Detailed Description	591
12.48.2 Field Documentation	593
12.48.2.1 <code>state</code>	593
12.48.2.2 <code>config</code>	593
12.48.2.3 <code>errors</code>	593
12.48.2.4 <code>mutex</code>	593
12.49ICUConfig Struct Reference	593
12.49.1 Detailed Description	593
12.49.2 Field Documentation	594
12.49.2.1 <code>mode</code>	594
12.49.2.2 <code>frequency</code>	594
12.49.2.3 <code>width_cb</code>	595

12.49.2.4 period_cb	595
12.49.2.5 overflow_cb	595
12.50ICUDriver Struct Reference	595
12.50.1 Detailed Description	595
12.50.2 Field Documentation	596
12.50.2.1 state	596
12.50.2.2 config	596
12.51chibios_rt::InQueue Class Reference	596
12.51.1 Detailed Description	596
12.51.2 Constructor & Destructor Documentation	598
12.51.2.1 InQueue	598
12.51.3 Member Function Documentation	598
12.51.3.1 getFulll	598
12.51.3.2 getEmptyl	599
12.51.3.3 isEmptyl	599
12.51.3.4 isFulll	599
12.51.3.5 resetl	600
12.51.3.6 putl	600
12.51.3.7 get	601
12.51.3.8 getTimeout	601
12.51.3.9 readTimeout	602
12.52chibios_rt::InQueueBuffer< N > Class Template Reference	603
12.52.1 Detailed Description	603
12.52.2 Constructor & Destructor Documentation	604
12.52.2.1 InQueueBuffer	604
12.53intctx Struct Reference	605
12.53.1 Detailed Description	605
12.54IOBus Struct Reference	605
12.54.1 Detailed Description	605
12.54.2 Field Documentation	605
12.54.2.1 portid	605
12.54.2.2 mask	605
12.54.2.3 offset	605
12.55MACConfig Struct Reference	606
12.55.1 Detailed Description	606
12.55.2 Field Documentation	606
12.55.2.1 mac_address	606
12.56MACDriver Struct Reference	606
12.56.1 Detailed Description	606
12.56.2 Field Documentation	607

12.56.2.1 state	607
12.56.2.2 config	608
12.56.2.3 tdsem	608
12.56.2.4 rdsem	608
12.56.2.5 rdevent	608
12.57MACReceiveDescriptor Struct Reference	608
12.57.1 Detailed Description	608
12.57.2 Field Documentation	608
12.57.2.1 offset	608
12.57.2.2 size	608
12.58MACTransmitDescriptor Struct Reference	608
12.58.1 Detailed Description	608
12.58.2 Field Documentation	609
12.58.2.1 offset	609
12.58.2.2 size	609
12.59chibios_rt::Mailbox Class Reference	609
12.59.1 Detailed Description	609
12.59.2 Constructor & Destructor Documentation	611
12.59.2.1 Mailbox	611
12.59.3 Member Function Documentation	611
12.59.3.1 reset	611
12.59.3.2 post	612
12.59.3.3 postS	612
12.59.3.4 postl	613
12.59.3.5 postAhead	614
12.59.3.6 postAheadS	614
12.59.3.7 postAheadl	615
12.59.3.8 fetch	616
12.59.3.9 fetchS	616
12.59.3.10 fetchl	617
12.59.3.11 getFreeCountl	618
12.59.3.12 getUsedCountl	618
12.59.4 Field Documentation	618
12.59.4.1 mb	618
12.60Mailbox Struct Reference	618
12.60.1 Detailed Description	618
12.60.2 Field Documentation	619
12.60.2.1 mb_buffer	620
12.60.2.2 mb_top	620
12.60.2.3 mb_wptr	620

12.60.2.4 mb_rdptr	620
12.60.2.5 mb_fullsem	620
12.60.2.6 mb_emptysem	620
12.61 chibios_rt::MailboxBuffer< N > Class Template Reference	620
12.61.1 Detailed Description	620
12.61.2 Constructor & Destructor Documentation	622
12.61.2.1 MailboxBuffer	622
12.62 memory_heap Struct Reference	622
12.62.1 Detailed Description	622
12.62.2 Field Documentation	624
12.62.2.1 h_provider	624
12.62.2.2 h_free	624
12.62.2.3 h_mtx	624
12.63 MemoryPool Struct Reference	624
12.63.1 Detailed Description	624
12.63.2 Field Documentation	625
12.63.2.1 mp_next	625
12.63.2.2 mp_object_size	625
12.63.2.3 mp_provider	625
12.64 chibios_rt::MemoryPool Class Reference	625
12.64.1 Detailed Description	625
12.64.2 Constructor & Destructor Documentation	626
12.64.2.1 MemoryPool	626
12.64.2.2 MemoryPool	627
12.64.3 Member Function Documentation	627
12.64.3.1 loadArray	627
12.64.3.2 alloc1	628
12.64.3.3 alloc	629
12.64.3.4 free	629
12.64.3.5 freel	630
12.64.4 Field Documentation	630
12.64.4.1 pool	630
12.65 MemoryStream Struct Reference	630
12.65.1 Detailed Description	631
12.65.2 Field Documentation	632
12.65.2.1 vmt	632
12.66 MemStreamVMT Struct Reference	632
12.66.1 Detailed Description	632
12.67 MMCConfig Struct Reference	632
12.67.1 Detailed Description	632

12.67.2 Field Documentation	634
12.67.2.1 spip	634
12.67.2.2 lscfg	634
12.67.2.3 hscfg	634
12.68 MMCDriver Struct Reference	634
12.68.1 Detailed Description	634
12.68.2 Field Documentation	636
12.68.2.1 vmt	636
12.68.2.2 config	637
12.69 MMCDriverVMT Struct Reference	637
12.69.1 Detailed Description	637
12.70 MMCSDBlockDevice Struct Reference	638
12.70.1 Detailed Description	638
12.70.2 Field Documentation	640
12.70.2.1 vmt	640
12.71 MMCSDBlockDeviceVMT Struct Reference	640
12.71.1 Detailed Description	640
12.72 Mutex Struct Reference	641
12.72.1 Detailed Description	641
12.72.2 Field Documentation	643
12.72.2.1 m_queue	643
12.72.2.2 m_owner	643
12.72.2.3 m_next	643
12.73 chibios_rt::Mutex Class Reference	643
12.73.1 Detailed Description	643
12.73.2 Constructor & Destructor Documentation	644
12.73.2.1 Mutex	644
12.73.3 Member Function Documentation	644
12.73.3.1 tryLock	644
12.73.3.2 tryLockS	645
12.73.3.3 lock	646
12.73.3.4 lockS	646
12.73.4 Field Documentation	647
12.73.4.1 mutex	647
12.74 chibios_rt::ObjectsPool< T, N > Class Template Reference	647
12.74.1 Detailed Description	647
12.74.2 Constructor & Destructor Documentation	649
12.74.2.1 ObjectsPool	649
12.75 chibios_rt::OutQueue Class Reference	649
12.75.1 Detailed Description	649

12.75.2 Constructor & Destructor Documentation	651
12.75.2.1 OutQueue	651
12.75.3 Member Function Documentation	651
12.75.3.1 getFulll	651
12.75.3.2 getEmptyl	652
12.75.3.3 isEmptyl	652
12.75.3.4 isFulll	652
12.75.3.5 resetl	653
12.75.3.6 put	653
12.75.3.7 putTimeout	653
12.75.3.8 getl	654
12.75.3.9 writeTimeout	655
12.76 chibios_rt::OutQueueBuffer< N > Class Template Reference	655
12.76.1 Detailed Description	656
12.76.2 Constructor & Destructor Documentation	657
12.76.2.1 OutQueueBuffer	657
12.77 PALConfig Struct Reference	658
12.77.1 Detailed Description	658
12.78 pool_header Struct Reference	658
12.78.1 Detailed Description	658
12.78.2 Field Documentation	658
12.78.2.1 ph_next	658
12.79 PWMChannelConfig Struct Reference	659
12.79.1 Detailed Description	659
12.79.2 Field Documentation	660
12.79.2.1 mode	660
12.79.2.2 callback	660
12.80 PWMConfig Struct Reference	660
12.80.1 Detailed Description	660
12.80.2 Field Documentation	661
12.80.2.1 frequency	661
12.80.2.2 period	662
12.80.2.3 callback	662
12.80.2.4 channels	662
12.81 PWDriver Struct Reference	662
12.81.1 Detailed Description	662
12.81.2 Field Documentation	663
12.81.2.1 state	663
12.81.2.2 config	663
12.81.2.3 period	664

12.82 ReadyList Struct Reference	664
12.82.1 Detailed Description	664
12.82.2 Field Documentation	666
12.82.2.1 r_queue	666
12.82.2.2 r_prio	666
12.82.2.3 r_ctx	666
12.82.2.4 r_newer	666
12.82.2.5 r_older	666
12.82.2.6 r_current	666
12.83 SDCCConfig Struct Reference	666
12.83.1 Detailed Description	666
12.84 SDCDriver Struct Reference	667
12.84.1 Detailed Description	667
12.84.2 Field Documentation	668
12.84.2.1 vmt	668
12.84.2.2 config	668
12.84.2.3 cardmode	668
12.84.2.4 errors	668
12.84.2.5 rca	668
12.85 SDCDriverVMT Struct Reference	668
12.85.1 Detailed Description	668
12.86 Semaphore Struct Reference	670
12.86.1 Detailed Description	670
12.86.2 Field Documentation	672
12.86.2.1 s_queue	672
12.86.2.2 s_cnt	672
12.87 SerialConfig Struct Reference	672
12.87.1 Detailed Description	672
12.87.2 Field Documentation	672
12.87.2.1 sc_speed	672
12.88 SerialDriver Struct Reference	672
12.88.1 Detailed Description	672
12.88.2 Field Documentation	674
12.88.2.1 vmt	674
12.89 SerialDriverVMT Struct Reference	674
12.89.1 Detailed Description	674
12.90 SerialUSBConfig Struct Reference	676
12.90.1 Detailed Description	676
12.90.2 Field Documentation	677
12.90.2.1 usbp	677

12.90.2.2 bulk_in	677
12.90.2.3 bulk_out	677
12.90.2.4 int_in	677
12.91 SerialUSBDriver Struct Reference	677
12.91.1 Detailed Description	677
12.91.2 Field Documentation	679
12.91.2.1 vmt	679
12.92 SerialUSBDriverVMT Struct Reference	679
12.92.1 Detailed Description	679
12.93 ShellCommand Struct Reference	681
12.93.1 Detailed Description	681
12.93.2 Field Documentation	681
12.93.2.1 sc_name	681
12.93.2.2 sc_function	681
12.94 ShellConfig Struct Reference	682
12.94.1 Detailed Description	682
12.94.2 Field Documentation	682
12.94.2.1 sc_channel	682
12.94.2.2 sc_commands	682
12.95 SPIConfig Struct Reference	683
12.95.1 Detailed Description	683
12.95.2 Field Documentation	685
12.95.2.1 end_cb	685
12.96 SPIDriver Struct Reference	685
12.96.1 Detailed Description	685
12.96.2 Field Documentation	687
12.96.2.1 state	687
12.96.2.2 config	687
12.96.2.3 thread	687
12.96.2.4 mutex	687
12.97 chibios_rt::System Class Reference	687
12.97.1 Detailed Description	687
12.97.2 Member Function Documentation	688
12.97.2.1 init	688
12.97.2.2 lock	689
12.97.2.3 unlock	689
12.97.2.4 lockFromIlsr	689
12.97.2.5 unlockFromIlsr	689
12.97.2.6 getTime	689
12.97.2.7 isTimeWithin	690

12.98testcase Struct Reference	690
12.98.1 Detailed Description	690
12.98.2 Field Documentation	690
12.98.2.1 name	690
12.98.2.2 setup	691
12.98.2.3 teardown	691
12.98.2.4 execute	691
12.99Thread Struct Reference	691
12.99.1 Detailed Description	691
12.99.2 Field Documentation	695
12.99.2.1 p_next	695
12.99.2.2 p_prev	695
12.99.2.3 p_prio	695
12.99.2.4 p_ctx	695
12.99.2.5 p_newer	695
12.99.2.6 p_older	695
12.99.2.7 p_name	695
12.99.2.8 p_stklimit	695
12.99.2.9 p_state	695
12.99.2.10p_flags	695
12.99.2.11p_refs	695
12.99.2.12p_preempt	696
12.99.2.13p_time	696
12.99.2.14rdymsg	696
12.99.2.15exitcode	696
12.99.2.16wtobjp	696
12.99.2.17ewmask	696
12.99.2.18p_waiting	696
12.99.2.19p_msgqueue	697
12.99.2.20p_msg	697
12.99.2.21p_pending	697
12.99.2.22p_mtxlist	697
12.99.2.23p_realprio	697
12.99.2.24p_mpool	697
12.100hibios_rt::ThreadReference Class Reference	697
12.100.1Detailed Description	697
12.100.2Constructor & Destructor Documentation	700
12.100.2.1ThreadReference	700
12.100.3Member Function Documentation	700
12.100.3.1stop	700

12.100.3.2 suspend	701
12.100.3.3 suspendS	701
12.100.3.4 resume	702
12.100.3.5 resumel	702
12.100.3.6 requestTerminate	703
12.100.3.7 wait	703
12.100.3.8 sendMessage	704
12.100.3.9 sPendingMessage	705
12.100.3.10 getMessage	705
12.100.3.11 releaseMessage	705
12.100.3.12 signalEvents	706
12.100.3.13 signalEventsl	706
12.100.4 Field Documentation	707
12.100.4.1 thread_ref	707
12.101 ThreadsList Struct Reference	707
12.101.1 Detailed Description	707
12.101.2 Field Documentation	710
12.101.2.1 p_next	710
12.102 ThreadsQueue Struct Reference	710
12.102.1 Detailed Description	710
12.102.2 Field Documentation	712
12.102.2.1 p_next	713
12.102.2.2 p_prev	713
12.103 TimeMeasurement Struct Reference	713
12.103.1 Detailed Description	713
12.103.2 Field Documentation	713
12.103.2.1 start	713
12.103.2.2 stop	713
12.103.2.3 last	713
12.103.2.4 worst	713
12.103.2.5 best	714
12.104 hibios_rt::Timer Class Reference	714
12.104.1 Detailed Description	714
12.104.2 Member Function Documentation	715
12.104.2.1 setl	715
12.104.2.2 resetl	715
12.104.2.3 sArmedl	716
12.104.3 Field Documentation	716
12.104.3.1 timer_ref	716
12.105 JARTConfig Struct Reference	716

12.105.1Detailed Description	716
12.105.2Field Documentation	717
12.105.2.1txend1_cb	717
12.105.2.2xend2_cb	717
12.105.2.3xend_cb	718
12.105.2.4xchar_cb	718
12.105.2.5xerr_cb	718
12.106JARTDriver Struct Reference	718
12.106.1Detailed Description	718
12.106.2Field Documentation	719
12.106.2.1state	719
12.106.2.2xstate	719
12.106.2.3xstate	719
12.106.2.4config	719
12.107USBConfig Struct Reference	719
12.107.1Detailed Description	719
12.107.2Field Documentation	721
12.107.2.1event_cb	721
12.107.2.2get_descriptor_cb	721
12.107.2.3requests_hook_cb	721
12.107.2.4sof_cb	721
12.108ISBDescriptor Struct Reference	721
12.108.1Detailed Description	721
12.108.2Field Documentation	721
12.108.2.1ud_size	721
12.108.2.2ud_string	722
12.109ISBDriver Struct Reference	722
12.109.1Detailed Description	722
12.109.2Field Documentation	723
12.109.2.1state	723
12.109.2.2config	723
12.109.2.3transmitting	723
12.109.2.4receiving	723
12.109.2.5epc	723
12.109.2.6n_params	723
12.109.2.7out_params	723
12.109.2.8ep0state	724
12.109.2.9ep0next	724
12.109.2.10ep0n	724
12.109.2.11ep0endcb	724

12.109.2.1 <u>Setup</u>	724
12.109.2.1 <u>Status</u>	724
12.109.2.1 <u>Address</u>	724
12.109.2.1 <u>Configuration</u>	724
12.110 <u>USBEndpointConfig Struct Reference</u>	724
12.110.1 <u>Detailed Description</u>	724
12.110.2 <u>Field Documentation</u>	726
12.110.2.1 <u>ep_mode</u>	726
12.110.2.2 <u>setup_cb</u>	726
12.110.2.3 <u>n_cb</u>	726
12.110.2.4 <u>out_cb</u>	726
12.110.2.5 <u>n_maxsize</u>	726
12.110.2.6 <u>out_maxsize</u>	727
12.110.2.7 <u>n_state</u>	727
12.110.2.8 <u>out_state</u>	727
12.111 <u>USBInEndpointState Struct Reference</u>	727
12.111.1 <u>Detailed Description</u>	727
12.111.2 <u>Field Documentation</u>	729
12.111.2.1 <u>txqueued</u>	729
12.111.2.2 <u>xsize</u>	729
12.111.2.3 <u>xcnt</u>	729
12.111.2.4 <u>xbuf</u>	729
12.111.2.5 <u>xqueue</u>	729
12.112 <u>USBOutEndpointState Struct Reference</u>	729
12.112.1 <u>Detailed Description</u>	729
12.112.2 <u>Field Documentation</u>	731
12.112.2.1 <u>rxqueued</u>	731
12.112.2.2 <u>xsize</u>	731
12.112.2.3 <u>xcnt</u>	731
12.112.2.4 <u>xbuf</u>	731
12.112.2.5 <u>xqueue</u>	731
12.113 <u>VirtualTimer Struct Reference</u>	731
12.113.1 <u>Detailed Description</u>	731
12.113.2 <u>Field Documentation</u>	733
12.113.2.1 <u>vt_next</u>	733
12.113.2.2 <u>vt_prev</u>	733
12.113.2.3 <u>vt_time</u>	733
12.113.2.4 <u>vt_func</u>	733
12.113.2.5 <u>vt_par</u>	733
12.114 <u>TList Struct Reference</u>	733

12.114. Detailed Description	733
12.114.2 Field Documentation	735
12.114.2.1 vt_next	735
12.114.2.2 vt_prev	735
12.114.2.3 vt_time	735
12.114.2.4 vt_systime	735
13 File Documentation	736
13.1 adc.c File Reference	736
13.1.1 Detailed Description	736
13.2 adc.h File Reference	737
13.2.1 Detailed Description	737
13.3 adc_lld.c File Reference	738
13.3.1 Detailed Description	738
13.4 adc_lld.h File Reference	738
13.4.1 Detailed Description	738
13.5 can.c File Reference	739
13.5.1 Detailed Description	739
13.6 can.h File Reference	740
13.6.1 Detailed Description	740
13.7 can_lld.c File Reference	741
13.7.1 Detailed Description	741
13.8 can_lld.h File Reference	742
13.8.1 Detailed Description	742
13.9 ch.cpp File Reference	743
13.9.1 Detailed Description	743
13.10ch.h File Reference	743
13.10.1 Detailed Description	743
13.11ch.hpp File Reference	745
13.11.1 Detailed Description	745
13.12chbsem.h File Reference	746
13.12.1 Detailed Description	746
13.13chcond.c File Reference	746
13.13.1 Detailed Description	746
13.14chcond.h File Reference	747
13.14.1 Detailed Description	747
13.15chconf.h File Reference	748
13.15.1 Detailed Description	748
13.16chcore.c File Reference	750
13.16.1 Detailed Description	750

13.17chcore.h File Reference	750
13.17.1 Detailed Description	750
13.18chdebug.c File Reference	752
13.18.1 Detailed Description	752
13.19chdebug.h File Reference	753
13.19.1 Detailed Description	753
13.20chdynamic.c File Reference	754
13.20.1 Detailed Description	754
13.21chdynamic.h File Reference	754
13.21.1 Detailed Description	754
13.22chevents.c File Reference	754
13.22.1 Detailed Description	754
13.23chevents.h File Reference	755
13.23.1 Detailed Description	755
13.24chfiles.h File Reference	757
13.24.1 Detailed Description	757
13.25chheap.c File Reference	757
13.25.1 Detailed Description	757
13.26chheap.h File Reference	758
13.26.1 Detailed Description	758
13.27chinline.h File Reference	758
13.27.1 Detailed Description	758
13.28chlists.c File Reference	758
13.28.1 Detailed Description	758
13.29chlists.h File Reference	759
13.29.1 Detailed Description	759
13.30chmboxes.c File Reference	760
13.30.1 Detailed Description	760
13.31chmboxes.h File Reference	760
13.31.1 Detailed Description	760
13.32chmemcore.c File Reference	761
13.32.1 Detailed Description	761
13.33chmemcore.h File Reference	762
13.33.1 Detailed Description	762
13.34chmpools.c File Reference	763
13.34.1 Detailed Description	763
13.35chmpools.h File Reference	763
13.35.1 Detailed Description	763
13.36chmsg.c File Reference	764
13.36.1 Detailed Description	764

13.37chmsg.h File Reference	764
13.37.1 Detailed Description	764
13.38chmtx.c File Reference	765
13.38.1 Detailed Description	765
13.39chmtx.h File Reference	765
13.39.1 Detailed Description	765
13.40chprintf.c File Reference	766
13.40.1 Detailed Description	766
13.41chprintf.h File Reference	767
13.41.1 Detailed Description	767
13.42chqueues.c File Reference	767
13.42.1 Detailed Description	767
13.43chqueues.h File Reference	768
13.43.1 Detailed Description	768
13.44chregistry.c File Reference	769
13.44.1 Detailed Description	769
13.45chregistry.h File Reference	770
13.45.1 Detailed Description	770
13.46chrtclib.c File Reference	770
13.46.1 Detailed Description	770
13.47chrtclib.h File Reference	771
13.47.1 Detailed Description	771
13.48chsched.c File Reference	771
13.48.1 Detailed Description	771
13.49chsched.h File Reference	772
13.49.1 Detailed Description	772
13.50chsem.c File Reference	774
13.50.1 Detailed Description	774
13.51chsem.h File Reference	774
13.51.1 Detailed Description	774
13.52chstreams.h File Reference	775
13.52.1 Detailed Description	775
13.53chsys.c File Reference	776
13.53.1 Detailed Description	776
13.54chsys.h File Reference	776
13.54.1 Detailed Description	776
13.55cthreads.c File Reference	777
13.55.1 Detailed Description	777
13.56cthreads.h File Reference	778
13.56.1 Detailed Description	778

13.57ctypes.h File Reference	780
13.57.1 Detailed Description	780
13.58chvt.c File Reference	781
13.58.1 Detailed Description	781
13.59chvt.h File Reference	781
13.59.1 Detailed Description	781
13.60evtimer.c File Reference	783
13.60.1 Detailed Description	783
13.61evtimer.h File Reference	783
13.61.1 Detailed Description	783
13.62ext.c File Reference	783
13.62.1 Detailed Description	783
13.63ext.h File Reference	784
13.63.1 Detailed Description	784
13.64ext_lld.c File Reference	785
13.64.1 Detailed Description	785
13.65ext_lld.h File Reference	786
13.65.1 Detailed Description	786
13.66gpt.c File Reference	786
13.66.1 Detailed Description	786
13.67gpt.h File Reference	787
13.67.1 Detailed Description	787
13.68gpt_lld.c File Reference	788
13.68.1 Detailed Description	788
13.69gpt_lld.h File Reference	789
13.69.1 Detailed Description	789
13.70hal.c File Reference	790
13.70.1 Detailed Description	790
13.71hal.h File Reference	790
13.71.1 Detailed Description	790
13.72hal_lld.c File Reference	791
13.72.1 Detailed Description	791
13.73hal_lld.h File Reference	791
13.73.1 Detailed Description	791
13.74halconf.h File Reference	792
13.74.1 Detailed Description	792
13.75i2c.c File Reference	794
13.75.1 Detailed Description	794
13.76i2c.h File Reference	794
13.76.1 Detailed Description	794

13.77i2c_lld.c File Reference	796
13.77.1 Detailed Description	796
13.78i2c_lld.h File Reference	796
13.78.1 Detailed Description	796
13.79icu.c File Reference	797
13.79.1 Detailed Description	797
13.80icu.h File Reference	798
13.80.1 Detailed Description	798
13.81icu_lld.c File Reference	799
13.81.1 Detailed Description	799
13.82icu_lld.h File Reference	799
13.82.1 Detailed Description	799
13.83io_block.h File Reference	800
13.83.1 Detailed Description	800
13.84io_channel.h File Reference	801
13.84.1 Detailed Description	801
13.85mac.c File Reference	802
13.85.1 Detailed Description	802
13.86mac.h File Reference	803
13.86.1 Detailed Description	803
13.87mac_lld.c File Reference	804
13.87.1 Detailed Description	804
13.88mac_lld.h File Reference	805
13.88.1 Detailed Description	805
13.89memstreams.c File Reference	806
13.89.1 Detailed Description	806
13.90memstreams.h File Reference	806
13.90.1 Detailed Description	806
13.91mmc_spi.c File Reference	807
13.91.1 Detailed Description	807
13.92mmc_spi.h File Reference	808
13.92.1 Detailed Description	808
13.93mmcsd.c File Reference	809
13.93.1 Detailed Description	809
13.94mmcsd.h File Reference	809
13.94.1 Detailed Description	809
13.95pal.c File Reference	812
13.95.1 Detailed Description	812
13.96pal.h File Reference	812
13.96.1 Detailed Description	812

13.97pal_lld.c File Reference	814
13.97.1 Detailed Description	814
13.98pal_lld.h File Reference	814
13.98.1 Detailed Description	814
13.99pwm.c File Reference	815
13.99.1 Detailed Description	815
13.10pwm.h File Reference	816
13.100. Detailed Description	816
13.10pwm_lld.c File Reference	817
13.101. Detailed Description	817
13.10pwm_lld.h File Reference	818
13.102. Detailed Description	818
13.10tc.c File Reference	819
13.103. Detailed Description	819
13.10tc.h File Reference	819
13.104. Detailed Description	819
13.10dc.c File Reference	820
13.105. Detailed Description	820
13.10dc.h File Reference	821
13.106. Detailed Description	821
13.10dc_lld.c File Reference	822
13.107. Detailed Description	822
13.10dc_lld.h File Reference	823
13.108. Detailed Description	823
13.10serial.c File Reference	825
13.109. Detailed Description	825
13.11serial.h File Reference	825
13.110. Detailed Description	825
13.11serial_lld.c File Reference	827
13.111. Detailed Description	827
13.11serial_lld.h File Reference	827
13.112. Detailed Description	827
13.11serial_usb.c File Reference	828
13.113. Detailed Description	828
13.11serial_usb.h File Reference	829
13.114. Detailed Description	829
13.11shell.c File Reference	830
13.115. Detailed Description	830
13.11shell.h File Reference	831
13.116. Detailed Description	831

13.11 spi .c File Reference	832
13.117.Detailed Description	832
13.11 spi .h File Reference	833
13.118.Detailed Description	833
13.11 spi_lld .c File Reference	834
13.119.Detailed Description	834
13.12 spi_lld .h File Reference	835
13.120.Detailed Description	835
13.12 test .c File Reference	836
13.121.Detailed Description	836
13.12 test .h File Reference	837
13.122.Detailed Description	837
13.12 testbmk .c File Reference	838
13.123.Detailed Description	838
13.123.2variable Documentation	838
13.123.2.1patternbmk	838
13.12 testbmk .h File Reference	838
13.124.Detailed Description	838
13.124.2variable Documentation	839
13.124.2.1patternbmk	839
13.12 testdyn .c File Reference	839
13.125.Detailed Description	839
13.125.2variable Documentation	839
13.125.2.1patterndyn	839
13.12 testdyn .h File Reference	839
13.126.Detailed Description	839
13.126.2variable Documentation	840
13.126.2.1patterndyn	840
13.12 testevt .c File Reference	840
13.127.Detailed Description	840
13.127.2variable Documentation	840
13.127.2.1patternevt	840
13.12 testevt .h File Reference	840
13.128.Detailed Description	840
13.128.2variable Documentation	840
13.128.2.1patternevt	840
13.12 testheap .c File Reference	841
13.129.Detailed Description	841
13.129.2variable Documentation	841
13.129.2.1patternheap	841

13.130estheap.h File Reference	841
13.130.1Detailed Description	841
13.130.2Variable Documentation	841
13.130.2.1patternheap	841
13.131estmbox.c File Reference	841
13.131.1Detailed Description	841
13.131.2Variable Documentation	842
13.131.2.1patternmbox	842
13.132estmbox.h File Reference	842
13.132.1Detailed Description	842
13.132.2Variable Documentation	842
13.132.2.1patternmbox	842
13.133estmsg.c File Reference	842
13.133.1Detailed Description	842
13.133.2Variable Documentation	843
13.133.2.1patternmsg	843
13.134estmsg.h File Reference	843
13.134.1Detailed Description	843
13.134.2Variable Documentation	843
13.134.2.1patternmsg	843
13.135estmtx.c File Reference	843
13.135.1Detailed Description	843
13.135.2Variable Documentation	843
13.135.2.1patternmtx	843
13.136estmtx.h File Reference	844
13.136.1Detailed Description	844
13.136.2Variable Documentation	844
13.136.2.1patternmtx	844
13.137estpools.c File Reference	844
13.137.1Detailed Description	844
13.138estpools.h File Reference	844
13.138.1Detailed Description	844
13.139estqueues.c File Reference	844
13.139.1Detailed Description	845
13.139.2Variable Documentation	845
13.139.2.1patternqueues	845
13.140estqueues.h File Reference	845
13.140.1Detailed Description	845
13.140.2Variable Documentation	845
13.140.2.1patternqueues	845

13.14 testsem.c File Reference	845
13.141.1Detailed Description	845
13.141.2Variable Documentation	846
13.141.2.1patternsem	846
13.14 testsem.h File Reference	846
13.142.1Detailed Description	846
13.142.2Variable Documentation	846
13.142.2.1patternsem	846
13.14 testthd.c File Reference	846
13.143.1Detailed Description	846
13.143.2Variable Documentation	847
13.143.2.1patternthd	847
13.14 testthd.h File Reference	847
13.144.1Detailed Description	847
13.144.2Variable Documentation	847
13.144.2.1patternthd	847
13.14 m.c File Reference	847
13.145.1Detailed Description	847
13.14 m.h File Reference	847
13.146.1Detailed Description	847
13.14 uart.c File Reference	848
13.147.1Detailed Description	848
13.14 uart.h File Reference	849
13.148.1Detailed Description	849
13.14 uart_lld.c File Reference	850
13.149.1Detailed Description	850
13.15 uart_lld.h File Reference	850
13.150.1Detailed Description	850
13.15 usb.c File Reference	852
13.151.1Detailed Description	852
13.15 usb.h File Reference	853
13.152.1Detailed Description	853
13.15 usb_lld.c File Reference	856
13.153.1Detailed Description	856
13.153.2Variable Documentation	857
13.153.2.1in	857
13.153.2.2out	857
13.15 usb_lld.h File Reference	857
13.154.1Detailed Description	857

Chapter 1

ChibiOS/RT

1.1 Copyright

Copyright (C) 2006..2013 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 and HAL API.

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.

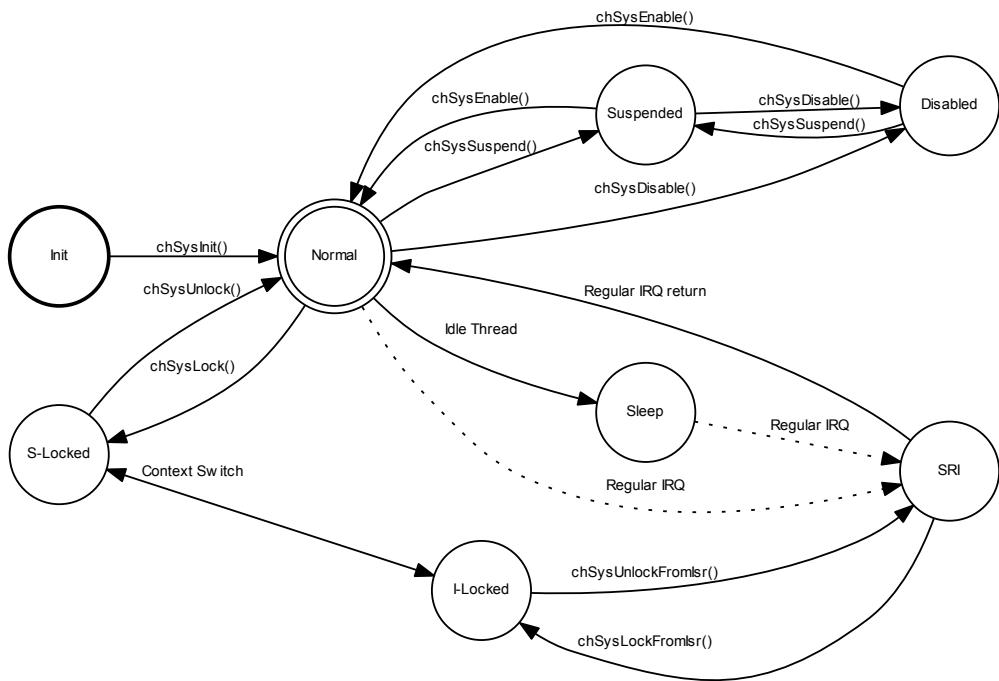
The mapping of the above logical classes into physical interrupts priorities is, of course, port dependent. See the documentation of the various ports for details.

2.4 System States

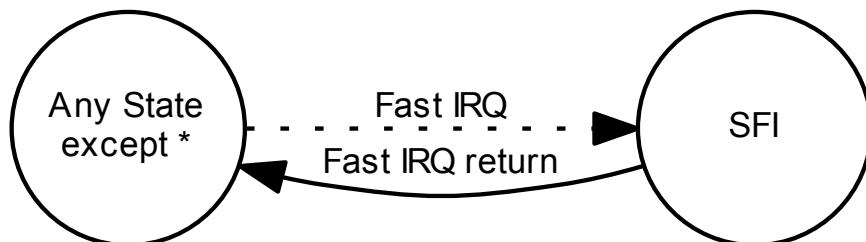
When using ChibiOS/RT the system can be in one of the following logical operating states:

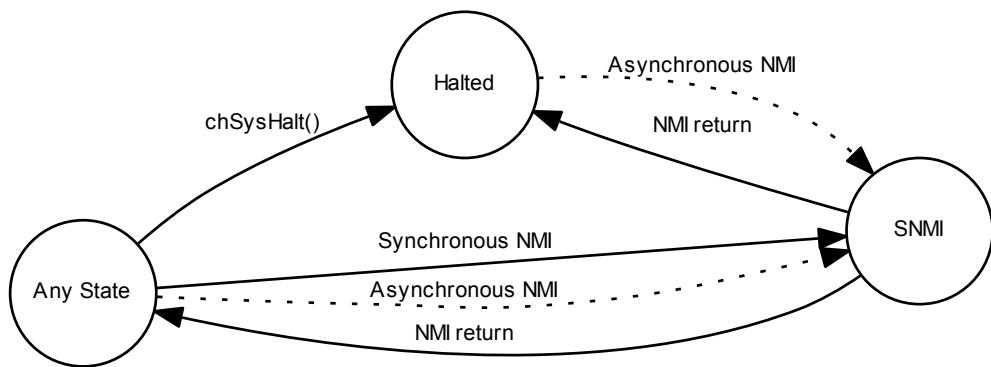
- **Init.** When the system is in this state all the maskable interrupt sources are disabled. In this state it is not possible to use any system API except `chSysInit()`. This state is entered after a physical reset.
- **Normal.** All the interrupt sources are enabled and the system APIs are accessible, 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:



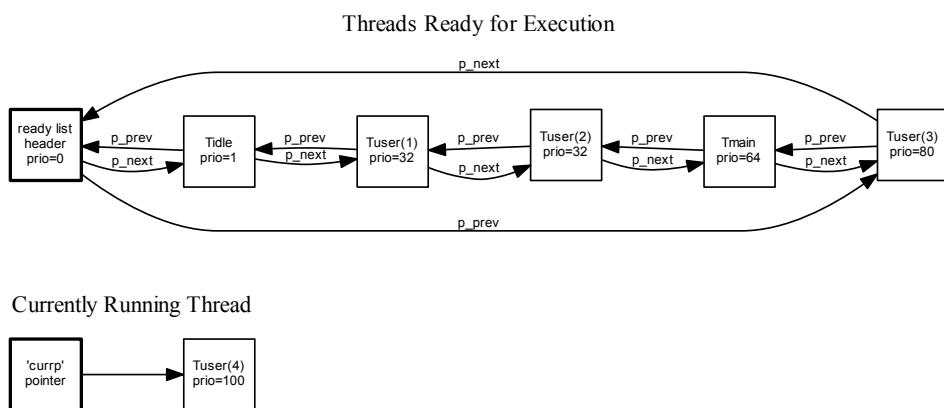


Attention

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

2.5 Scheduling

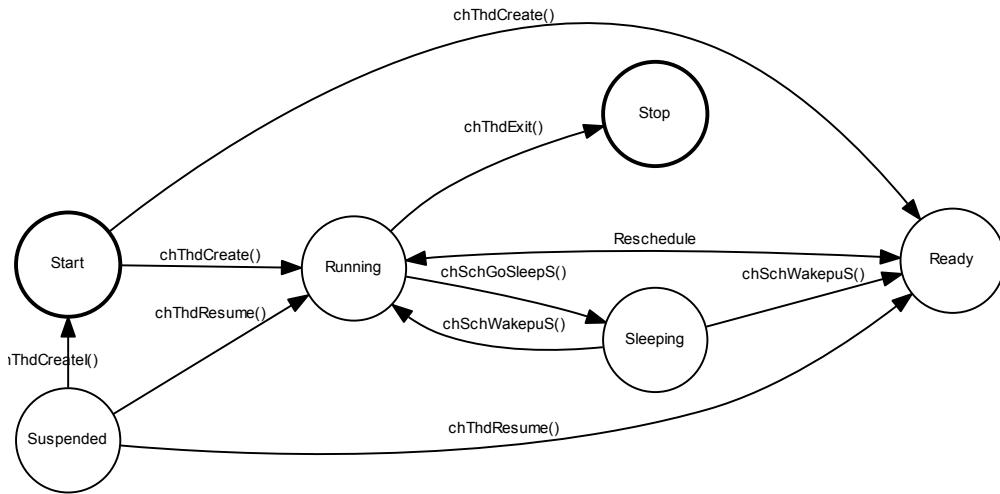
The strategy is very simple the currently ready thread with the highest priority is executed. If more than one thread with equal priority are eligible for execution then they are executed in a round-robin way, the CPU time slice constant is configurable. The ready list is a double linked list of threads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

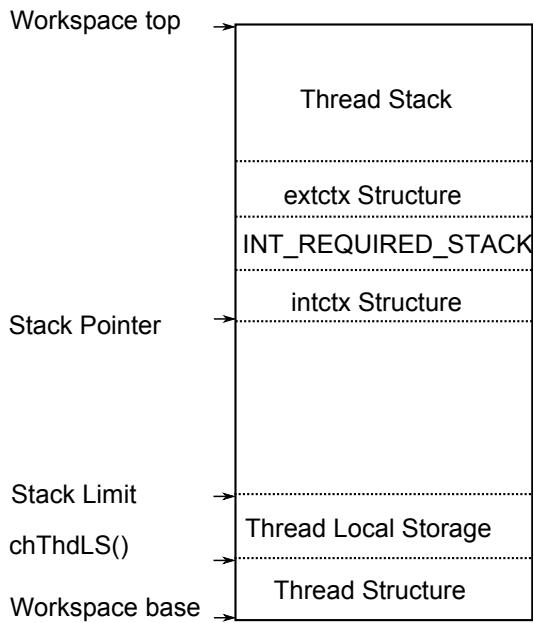
Priorities in ChibiOS/RT are a contiguous numerical range but the initial and final values are not enforced.

The following table describes the various priority boundaries (from lowest to highest):

- IDLEPRIO, 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.
- LOWPRIO, the lowest priority level that can be assigned to an user thread.
- NORMALPRIO, this is the central priority level for user threads. It is advisable to assign priorities to threads as values relative to NORMALPRIO, as example NORMALPRIO-1 or NORMALPRIO+4, this ensures the portability of code should the numerical range change in future implementations.
- HIGHPRIO, the highest priority level that can be assigned to an user thread.
- ABSPRIO, absolute maximum software priority level, it can be higher than HIGHPRIO but the numerical values above HIGHPRIO up to ABSPRIO (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 from its stack. 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.

Strategy by Component

The OS components are tested in various modes depending on their importance:

- **Kernel.** The kernel code is subject to rigorous testing. The test suite aims to test **all** the kernel code and reach a code coverage as close to 100% as possible. In addition to the code coverage, the kernel code is tested for **functionality** and benchmarked for **speed** and **size** before each stable release. In addition to the code coverage and functional testing a **batch compilation test** is performed before each release, the kernel is compiled by alternatively enabling and disabling all the various configuration options, the kernel code is expected to compile without errors nor warnings and execute the test suite without failures (a specific simulator is used for this execution test, it is done automatically by a script because the entire sequence can take hours).

All the tests results are included as reports in the OS distribution under `./docs/reports`.

- **Ports.** The port code is tested by executing the kernel test suite on the target hardware. A port is validated only if it passes all the tests. Speed and size benchmarks for all the supported architectures are performed, both size and speed regressions are **monitored**.
- **HAL.** The HAL high level code and device drivers implementations are tested through specific test applications under `./testhal`.
- **Various.** The miscellaneous code is tested by use in the various demos.
- **External Code.** Not tested, external libraries or components are used as-is or with minor patching where required, problems are usually reported upstream.

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](#)
- [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_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_USE_DYNAMIC`
- `CH_USE_HEAP`
- `CH_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_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_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_USE_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_USE_MUTEXES
- CH_USE_CONDVARNS
- 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

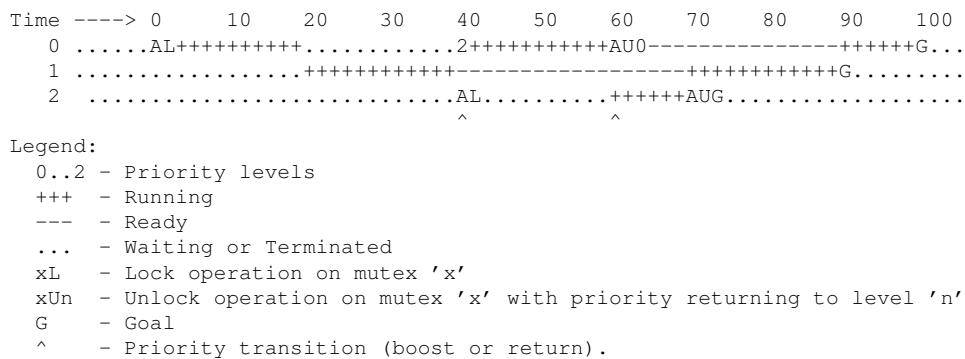
Description

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:



3.5.3 Priority inheritance, complex case

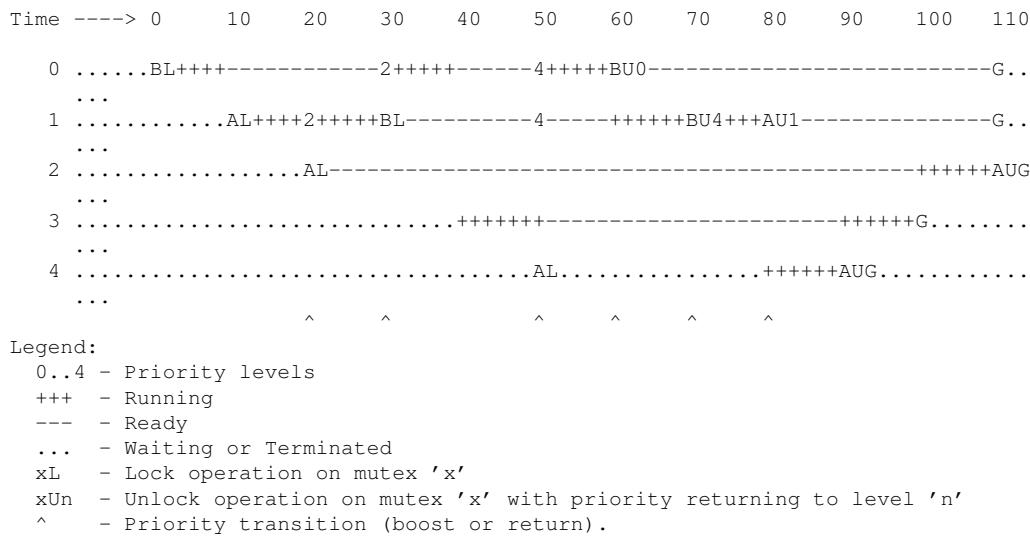
Description

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:



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_USE_EVENTS
- CH_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 meet its testing objectives as well.

Preconditions

The module requires the following kernel options:

- CH_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_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_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_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 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

3.11.1 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.

3.11.2 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.

3.11.3 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.

3.11.4 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.

3.11.5 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.

3.11.6 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.

3.11.7 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.

3.11.8 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.

3.11.9 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.

3.11.10 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.

3.11.11 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.

3.11.12 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.

3.11.13 RAM Footprint**Description**

The memory size of the various kernel objects is printed.

Chapter 4

Deprecated List

Global `sdGetWouldBlock(sdp)`

Global `sdPutWouldBlock(sdp)`

Chapter 5

Module Index

5.1 Modules

Here is a list of all modules:

Kernel	37
Version Numbers and Identification	37
Configuration	39
Types	49
Base Kernel Services	51
System Management	51
Scheduler	59
Threads	69
Time and Virtual Timers	83
Synchronization	90
Counting Semaphores	90
Binary Semaphores	100
Mutexes	105
Condition Variables	113
Event Flags	120
Synchronous Messages	133
Mailboxes	137
I/O Queues	148
Memory Management	163
Core Memory Manager	163
Heaps	166
Memory Pools	169
Dynamic Threads	175
Streams and Files	178
Abstract Sequential Streams	178
Abstract File Streams	181
Registry	184
Debug	187
Internals	196
Port	200
HAL	471
ADC Driver	205
CAN Driver	221
EXT Driver	233
GPT Driver	243
HAL Driver	254
I2C Driver	263

I2S Driver	274
ICU Driver	274
Abstract I/O Block Device	285
Abstract I/O Channel	291
MAC Driver	297
MMC over SPI Driver	310
MMC/SD Block Device	322
PAL Driver	327
PWM Driver	346
RTC Driver	359
SDC Driver	363
Serial Driver	381
Serial over USB Driver	394
SPI Driver	401
Time Measurement Driver	420
UART Driver	422
USB Driver	437
Configuration	473
Various	479
C++ Wrapper	479
Memory Streams	479
Periodic Events Timer	480
Command Shell	482
RTC time conversion utilities	486
System formatted print	490
Test Runtime	492
External Components	497

Chapter 6

Namespace Index

6.1 Namespace List

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

chibios_rt (ChibiOS kernel-related classes and interfaces)	498
---	-----

Chapter 7

Class Hierarchy Index

7.1 Class Hierarchy

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

ADCConfig	500
ADCConversionGroup	500
ADCDriver	502
BaseBlockDevice	508
MMCSDBlockDevice	638
MMCDriver	634
BaseBlockDeviceVMT	510
MMCSDBlockDeviceVMT	640
MMCDriverVMT	637
SDCDriverVMT	668
BaseSequentialStream	516
BaseChannel	510
BaseAsynchronousChannel	504
SerialDriver	672
SerialUSBDriver	677
BaseFileStream	514
MemoryStream	630
chibios_rt::BaseSequentialStreamInterface	518
BaseSequentialStreamVMT	520
BaseChannelVMT	512
BaseAsynchronousChannelVMT	506
SerialDriverVMT	674
SerialUSBDriverVMT	679
BaseFileStreamVMT	515
chibios_rt::BinarySemaphore	540
BlockDeviceInfo	544
CANConfig	544
CANDriver	545
CANFilter	547
CANRxFrame	547
CANTxFrame	548
cdc_linecoding_t	549
ch_swc_event_t	549
ch_trace_buffer_t	551
chdebug_t	553
CondVar	555
chibios_rt::CondVar	557

context	562
chibios_rt::Core	562
chibios_rt::CounterSemaphore	564
EventListener	572
EventSource	574
EvTimer	576
chibios_rt::EvtListener	576
chibios_rt::EvtSource	578
EXTChannelConfig	581
EXTConfig	582
extctx	583
EXTDriver	584
GenericQueue	585
GPTConfig	588
GPTDriver	589
heap_header	590
I2CConfig	591
I2CDriver	591
ICUConfig	593
ICUDriver	595
chibios_rt::InQueue	596
chibios_rt::InQueueBuffer< N >	603
intctx	605
IOBus	605
MACConfig	606
MACDriver	606
MACReceiveDescriptor	608
MACTransmitDescriptor	608
chibios_rt::Mailbox	609
chibios_rt::MailboxBuffer< N >	620
Mailbox	618
memory_heap	622
MemoryPool	624
chibios_rt::MemoryPool	625
chibios_rt::ObjectsPool< T, N >	647
MemStreamVMT	632
MMCConfig	632
Mutex	641
chibios_rt::Mutex	643
chibios_rt::OutQueue	649
chibios_rt::OutQueueBuffer< N >	655
PALConfig	658
pool_header	658
PWMChannelConfig	659
PWMConfig	660
PWMDriver	662
SDCConfig	666
SDCDriver	667
Semaphore	670
BinarySemaphore	538
SerialConfig	672
SerialUSBConfig	676
ShellCommand	681
ShellConfig	682
SPIConfig	683
SPIDriver	685
chibios_rt::System	687

testcase	690
chibios_rt::ThreadReference	697
chibios_rt::BaseThread	523
chibios_rt::BaseStaticThread< N >	520
ThreadsList	707
ThreadsQueue	710
ReadyList	664
Thread	691
TimeMeasurement	713
chibios_rt::Timer	714
UARTConfig	716
UARTDriver	718
USBConfig	719
USBDescriptor	721
USBDriver	722
USBEndpointConfig	724
USBInEndpointState	727
USBOutEndpointState	729
VTList	733
VirtualTimer	731

Chapter 8

Data Structure Index

8.1 Data Structures

Here are the data structures with brief descriptions:

<code>ADCConfig</code> (Driver configuration structure)	500
<code>ADCConversionGroup</code> (Conversion group configuration structure)	500
<code>ADCDriver</code> (Structure representing an ADC driver)	502
<code>BaseAsynchronousChannel</code> (Base asynchronous channel class)	504
<code>BaseAsynchronousChannelVMT</code> (<code>BaseAsynchronousChannel</code> virtual methods table)	506
<code>BaseBlockDevice</code> (Base block device class)	508
<code>BaseBlockDeviceVMT</code> (<code>BaseBlockDevice</code> virtual methods table)	510
<code>BaseChannel</code> (Base channel class)	510
<code>BaseChannelVMT</code> (<code>BaseChannel</code> virtual methods table)	512
<code>BaseFileStream</code> (Base file stream class)	514
<code>BaseFileStreamVMT</code> (<code>BaseFileStream</code> virtual methods table)	515
<code>BaseSequentialStream</code> (Base stream class)	516
<code>chibios_rt::BaseSequentialStreamInterface</code> (Interface of a <code>BaseSequentialStream</code>)	518
<code>BaseSequentialStreamVMT</code> (<code>BaseSequentialStream</code> virtual methods table)	520
<code>chibios_rt::BaseStaticThread< N ></code> (Static threads template class)	520
<code>chibios_rt::BaseThread</code> (Abstract base class for a ChibiOS/RT thread)	523
<code>BinarySemaphore</code> (Binary semaphore type)	538
<code>chibios_rt::BinarySemaphore</code> (Class encapsulating a binary semaphore)	540
<code>BlockDeviceInfo</code> (Block device info)	544
<code>CANConfig</code> (Driver configuration structure)	544
<code>CANDriver</code> (Structure representing an CAN driver)	545
<code>CANFilter</code> (CAN filter)	547
<code>CANRxFrame</code> (CAN received frame)	547
<code>CANTxFrame</code> (CAN transmission frame)	548
<code>cdc_linecoding_t</code> (Type of Line Coding structure)	549
<code>ch_swc_event_t</code> (Trace buffer record)	549
<code>ch_trace_buffer_t</code> (Trace buffer header)	551
<code>chdebug_t</code> (ChibiOS/RT memory signature record)	553
<code>CondVar</code> (<code>CondVar</code> structure)	555
<code>chibios_rt::CondVar</code> (Class encapsulating a conditional variable)	557
<code>context</code> (Platform dependent part of the <code>Thread</code> structure)	562
<code>chibios_rt::Core</code> (Class encapsulating the base system functionalities)	562
<code>chibios_rt::CounterSemaphore</code> (Class encapsulating a semaphore)	564
<code>EventListener</code> (Event Listener structure)	572
<code>EventSource</code> (Event Source structure)	574
<code>EvTimer</code> (Event timer structure)	576
<code>chibios_rt::EvtListener</code> (Class encapsulating an event listener)	576
<code>chibios_rt::EvtSource</code> (Class encapsulating an event source)	578

<code>EXTChannelConfig</code> (Channel configuration structure)	581
<code>EXTConfig</code> (Driver configuration structure)	582
<code>extctx</code> (Interrupt saved context)	583
<code>EXTDriver</code> (Structure representing an EXT driver)	584
<code>GenericQueue</code> (Generic I/O queue structure)	585
<code>GPTConfig</code> (Driver configuration structure)	588
<code>GPTDriver</code> (Structure representing a GPT driver)	589
<code>heap_header</code> (Memory heap block header)	590
<code>I2CConfig</code> (Driver configuration structure)	591
<code>I2CDriver</code> (Structure representing an I2C driver)	591
<code>ICUConfig</code> (Driver configuration structure)	593
<code>ICUDriver</code> (Structure representing an ICU driver)	595
<code>chibios_rt::InQueue</code> (Class encapsulating an input queue)	596
<code>chibios_rt::InQueueBuffer< N ></code> (Template class encapsulating an input queue and its buffer)	603
<code>intctx</code> (System saved context)	605
<code>IOBus</code> (I/O bus descriptor)	605
<code>MACConfig</code> (Driver configuration structure)	606
<code>MACDriver</code> (Structure representing a MAC driver)	606
<code>MACReceiveDescriptor</code> (Structure representing a receive descriptor)	608
<code>MACTransmitDescriptor</code> (Structure representing a transmit descriptor)	608
<code>chibios_rt::Mailbox</code> (Class encapsulating a mailbox)	609
<code>Mailbox</code> (Structure representing a mailbox object)	618
<code>chibios_rt::MailboxBuffer< N ></code> (Template class encapsulating a mailbox and its messages buffer)	620
<code>memory_heap</code> (Structure describing a memory heap)	622
<code>MemoryPool</code> (Memory pool descriptor)	624
<code>chibios_rt::MemoryPool</code> (Class encapsulating a mailbox)	625
<code>MemoryStream</code> (Memory stream object)	630
<code>MemStreamVMT</code> (<code>MemStream</code> virtual methods table)	632
<code>MMCConfig</code> (MMC/SD over SPI driver configuration structure)	632
<code>MMCDriver</code> (Structure representing a MMC/SD over SPI driver)	634
<code>MMCDriverVMT</code> (<code>MMCDriver</code> virtual methods table)	637
<code>MMCSDBlockDevice</code> (MCC/SD block device class)	638
<code>MMCSDBlockDeviceVMT</code> (<code>MMCSDBlockDevice</code> virtual methods table)	640
<code>Mutex</code> (<code>Mutex</code> structure)	641
<code>chibios_rt::Mutex</code> (Class encapsulating a mutex)	643
<code>chibios_rt::ObjectsPool< T, N ></code> (Template class encapsulating a memory pool and its elements)	647
<code>chibios_rt::OutQueue</code> (Class encapsulating an output queue)	649
<code>chibios_rt::OutQueueBuffer< N ></code> (Template class encapsulating an output queue and its buffer)	655
<code>PALConfig</code> (Generic I/O ports static initializer)	658
<code>pool_header</code> (Memory pool free object header)	658
<code>PWMChannelConfig</code> (PWM driver channel configuration structure)	659
<code>PWMConfig</code> (Driver configuration structure)	660
<code>PWMDriver</code> (Structure representing an PWM driver)	662
<code>ReadyList</code> (Ready list header)	664
<code>SDCConfig</code> (Driver configuration structure)	666
<code>SDCDriver</code> (Structure representing an SDC driver)	667
<code>SDCDriverVMT</code> (<code>SDCDriver</code> virtual methods table)	668
<code>Semaphore</code> (<code>Semaphore</code> structure)	670
<code>SerialConfig</code> (Generic Serial Driver configuration structure)	672
<code>SerialDriver</code> (Full duplex serial driver class)	672
<code>SerialDriverVMT</code> (<code>SerialDriver</code> virtual methods table)	674
<code>SerialUSBConfig</code> (Serial over USB Driver configuration structure)	676
<code>SerialUSBDriver</code> (Full duplex serial driver class)	677
<code>SerialUSBDriverVMT</code> (<code>SerialDriver</code> virtual methods table)	679
<code>ShellCommand</code> (Custom command entry type)	681
<code>ShellConfig</code> (Shell descriptor type)	682
<code>SPIConfig</code> (Driver configuration structure)	683
<code>SPIDriver</code> (Structure representing an SPI driver)	685
<code>chibios_rt::System</code> (Class encapsulating the base system functionalities)	687

testcase (Structure representing a test case)	690
Thread (Structure representing a thread)	691
chibios_rt::ThreadReference (Thread reference class)	697
ThreadsList (Generic threads single link list, it works like a stack)	707
ThreadsQueue (Generic threads bidirectional linked list header and element)	710
TimeMeasurement (Time Measurement structure)	713
chibios_rt::Timer (Timer class)	714
UARTConfig (Driver configuration structure)	716
UARTDriver (Structure representing an UART driver)	718
USBConfig (Type of an USB driver configuration structure)	719
USBDescriptor (Type of an USB descriptor)	721
USBDriver (Structure representing an USB driver)	722
USBEndpointConfig (Type of an USB endpoint configuration structure)	724
USBInEndpointState (Type of an IN endpoint state structure)	727
USBOutEndpointState (Type of an OUT endpoint state structure)	729
VirtualTimer (Virtual Timer descriptor structure)	731
VTList (Virtual timers list header)	733

Chapter 9

File Index

9.1 File List

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

<code>adc.c</code> (ADC Driver code)	736
<code>adc.h</code> (ADC Driver macros and structures)	737
<code>adc_lld.c</code> (ADC Driver subsystem low level driver source template)	738
<code>adc_lld.h</code> (ADC Driver subsystem low level driver header template)	738
<code>can.c</code> (CAN Driver code)	739
<code>can.h</code> (CAN Driver macros and structures)	740
<code>can_lld.c</code> (CAN Driver subsystem low level driver source template)	741
<code>can_lld.h</code> (CAN Driver subsystem low level driver header template)	742
<code>ch.cpp</code> (C++ wrapper code)	743
<code>ch.h</code> (ChibiOS/RT main include file)	743
<code>ch.hpp</code> (C++ wrapper classes and definitions)	745
<code>chbsem.h</code> (Binary semaphores structures and macros)	746
<code>chcond.c</code> (Condition Variables code)	746
<code>chcond.h</code> (Condition Variables macros and structures)	747
<code>chconf.h</code> (Configuration file template)	748
<code>chcore.c</code> (Port related template code)	750
<code>chcore.h</code> (Port related template macros and structures)	750
<code>chdebug.c</code> (ChibiOS/RT Debug code)	752
<code>chdebug.h</code> (Debug macros and structures)	753
<code>chdynamic.c</code> (Dynamic threads code)	754
<code>chdynamic.h</code> (Dynamic threads macros and structures)	754
<code>chevents.c</code> (Events code)	754
<code>chevents.h</code> (Events macros and structures)	755
<code>chfiles.h</code> (Data files)	757
<code>chheap.c</code> (Heaps code)	757
<code>chheap.h</code> (Heaps macros and structures)	758
<code>chinline.h</code> (Kernel inlined functions)	758
<code>chlists.c</code> (Thread queues/lists code)	758
<code>chlists.h</code> (Thread queues/lists macros and structures)	759
<code>chmboxes.c</code> (Mailboxes code)	760
<code>chmboxes.h</code> (Mailboxes macros and structures)	760
<code>chmemcore.c</code> (Core memory manager code)	761
<code>chmemcore.h</code> (Core memory manager macros and structures)	762
<code>chmempools.c</code> (Memory Pools code)	763
<code>chmempools.h</code> (Memory Pools macros and structures)	763
<code>chmsg.c</code> (Messages code)	764
<code>chmsg.h</code> (Messages macros and structures)	764
<code>chmtx.c</code> (Mutexes code)	765

chmtx.h (Mutexes macros and structures)	765
chprintf.c (Mini printf-like functionality)	766
chprintf.h (Mini printf-like functionality)	767
chqueues.c (I/O Queues code)	767
chqueues.h (I/O Queues macros and structures)	768
chregistry.c (Threads registry code)	769
chregistry.h (Threads registry macros and structures)	770
chrtclib.c (RTC time conversion utilities code)	770
chrtclib.h (RTC time conversion utilities header)	771
chsched.c (Scheduler code)	771
chsched.h (Scheduler macros and structures)	772
chsem.c (Semaphores code)	774
chsem.h (Semaphores macros and structures)	774
chstreams.h (Data streams)	775
chsys.c (System related code)	776
chsys.h (System related macros and structures)	776
chthreads.c (Threads code)	777
chthreads.h (Threads macros and structures)	778
ctypes.h (System types template)	780
chvt.c (Time and Virtual Timers related code)	781
chvt.h (Time macros and structures)	781
evtimer.c (Events Generator Timer code)	783
evtimer.h (Events Generator Timer structures and macros)	783
ext.c (EXT Driver code)	783
ext.h (EXT Driver macros and structures)	784
ext_lld.c (EXT Driver subsystem low level driver source template)	785
ext_lld.h (EXT Driver subsystem low level driver header template)	786
gpt.c (GPT Driver code)	786
gpt.h (GPT Driver macros and structures)	787
gpt_lld.c (GPT Driver subsystem low level driver source template)	788
gpt_lld.h (GPT Driver subsystem low level driver header template)	789
hal.c (HAL subsystem code)	790
hal.h (HAL subsystem header)	790
hal_lld.c (HAL Driver subsystem low level driver source template)	791
hal_lld.h (HAL subsystem low level driver header template)	791
halconf.h (HAL configuration header)	792
i2c.c (I2C Driver code)	794
i2c.h (I2C Driver macros and structures)	794
i2c_lld.c (I2C Driver subsystem low level driver source template)	796
i2c_lld.h (I2C Driver subsystem low level driver header template)	796
icu.c (ICU Driver code)	797
icu.h (ICU Driver macros and structures)	798
icu_lld.c (ICU Driver subsystem low level driver source template)	799
icu_lld.h (ICU Driver subsystem low level driver header template)	799
io_block.h (I/O block devices access)	800
io_channel.h (I/O channels access)	801
mac.c (MAC Driver code)	802
mac.h (MAC Driver macros and structures)	803
mac_lld.c (MAC Driver subsystem low level driver source template)	804
mac_lld.h (MAC Driver subsystem low level driver header template)	805
memstreams.c (Memory streams code)	806
memstreams.h (Memory streams structures and macros)	806
mmc_spi.c (MMC over SPI driver code)	807
mmc_spi.h (MMC over SPI driver header)	808
mmcsd.c (MMC/SD cards common code)	809
mmcsd.h (MMC/SD cards common header)	809
pal.c (I/O Ports Abstraction Layer code)	812
pal.h (I/O Ports Abstraction Layer macros, types and structures)	812
pal_lld.c (PAL subsystem low level driver template)	814

<code>pal_lld.h</code> (PAL subsystem low level driver header template)	814
<code>pwm.c</code> (PWM Driver code)	815
<code>pwm.h</code> (PWM Driver macros and structures)	816
<code>pwm_lld.c</code> (PWM Driver subsystem low level driver source template)	817
<code>pwm_lld.h</code> (PWM Driver subsystem low level driver header template)	818
<code>rtc.c</code> (RTC Driver code)	819
<code>rtc.h</code> (RTC Driver macros and structures)	819
<code>sdc.c</code> (SDC Driver code)	820
<code>sdc.h</code> (SDC Driver macros and structures)	821
<code>sdc_lld.c</code> (SDC Driver subsystem low level driver source template)	822
<code>sdc_lld.h</code> (SDC Driver subsystem low level driver header template)	823
<code>serial.c</code> (Serial Driver code)	825
<code>serial.h</code> (Serial Driver macros and structures)	825
<code>serial_lld.c</code> (Serial Driver subsystem low level driver source template)	827
<code>serial_lld.h</code> (Serial Driver subsystem low level driver header template)	827
<code>serial_usb.c</code> (Serial over USB Driver code)	828
<code>serial_usb.h</code> (Serial over USB Driver macros and structures)	829
<code>shell.c</code> (Simple CLI shell code)	830
<code>shell.h</code> (Simple CLI shell header)	831
<code>spi.c</code> (SPI Driver code)	832
<code>spi.h</code> (SPI Driver macros and structures)	833
<code>spi_lld.c</code> (SPI Driver subsystem low level driver source template)	834
<code>spi_lld.h</code> (SPI Driver subsystem low level driver header template)	835
<code>test.c</code> (Tests support code)	836
<code>test.h</code> (Tests support header)	837
<code>testbmk.c</code> (Kernel Benchmarks source file)	838
<code>testbmk.h</code> (Kernel Benchmarks header file)	838
<code>testdyn.c</code> (Dynamic thread APIs test source file)	839
<code>testdyn.h</code> (Dynamic thread APIs test header file)	839
<code>testevt.c</code> (Events test source file)	840
<code>testevt.h</code> (Events test header file)	840
<code>testheap.c</code> (Heap test source file)	841
<code>testheap.h</code> (Heap header file)	841
<code>testmbox.c</code> (Mailboxes test source file)	841
<code>testmbox.h</code> (Mailboxes header file)	842
<code>testmsg.c</code> (Messages test source file)	842
<code>testmsg.h</code> (Messages header file)	843
<code>testmtx.c</code> (Mutexes and CondVars test source file)	843
<code>testmtx.h</code> (Mutexes and CondVars test header file)	844
<code>testpools.c</code> (Memory Pools test source file)	844
<code>testpools.h</code> (Memory Pools test header file)	844
<code>testqueues.c</code> (I/O Queues test source file)	844
<code>testqueues.h</code> (I/O Queues test header file)	845
<code>testsem.c</code> (Semaphores test source file)	845
<code>testsem.h</code> (Semaphores test header file)	846
<code>testthd.c</code> (Threads and Scheduler test source file)	846
<code>testthd.h</code> (Threads and Scheduler test header file)	847
<code>tm.c</code> (Time Measurement driver code)	847
<code>tm.h</code> (Time Measurement driver header)	847
<code>uart.c</code> (UART Driver code)	848
<code>uart.h</code> (UART Driver macros and structures)	849
<code>uart_lld.c</code> (UART Driver subsystem low level driver source template)	850
<code>uart_lld.h</code> (UART Driver subsystem low level driver header template)	850
<code>usb.c</code> (USB Driver code)	852
<code>usb.h</code> (USB Driver macros and structures)	853
<code>usb_lld.c</code> (USB Driver subsystem low level driver source template)	856
<code>usb_lld.h</code> (USB Driver subsystem low level driver header template)	857

Chapter 10

Module Documentation

10.1 Kernel

10.1.1 Detailed Description

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

Modules

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

10.2 Version Numbers and Identification

10.2.1 Detailed Description

Kernel related info.

Functions

- void [_idle_thread](#) (void *p)
This function implements the idle thread infinite loop.

Kernel version

- #define [CH_KERNEL_MAJOR](#) 2
Kernel version major number.
- #define [CH_KERNEL_MINOR](#) 6

- `#define CH_KERNEL_PATCH 9`
Kernel version patch number.

Common constants

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

Defines

- `#define _CHIBIOS_RT_`
ChibiOS/RT identification macro.
- `#define CH_KERNEL_VERSION "2.6.9"`
Kernel version string.

10.2.2 Function Documentation

10.2.2.1 void _idle_thread (void * p)

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 *p* the thread parameter, unused in this scenario

Here is the call graph for this function:



10.2.3 Define Documentation

10.2.3.1 #define _CHIBIOS_RT_

ChibiOS/RT identification macro.

10.2.3.2 `#define CH_KERNEL_VERSION "2.6.9"`

Kernel version string.

10.2.3.3 `#define CH_KERNEL_MAJOR 2`

Kernel version major number.

10.2.3.4 `#define CH_KERNEL_MINOR 6`

Kernel version minor number.

10.2.3.5 `#define CH_KERNEL_PATCH 9`

Kernel version patch number.

10.2.3.6 `#define FALSE 0`

Generic 'false' boolean constant.

10.2.3.7 `#define TRUE (!FALSE)`

Generic 'true' boolean constant.

10.2.3.8 `#define CH_SUCCESS FALSE`

Generic success constant.

This constant is functionally equivalent to FALSE but more readable, it can be used as return value of all those functions returning a `bool_t` as a status indicator.

10.2.3.9 `#define CH_FAILED TRUE`

Generic failure constant.

This constant is functionally equivalent to TRUE but more readable, it can be used as return value of all those functions returning a `bool_t` as a status indicator.

10.3 Configuration

10.3.1 Detailed Description

Kernel related settings and hooks.

Kernel parameters and options

- `#define CH_FREQUENCY 1000`
System tick frequency.
- `#define CH_TIME_QUANTUM 20`
Round robin interval.
- `#define CH_MEMCORE_SIZE 0`

- `#define CH_NO_IDLE_THREAD FALSE`
Idle thread automatic spawn suppression.

Performance options

- `#define CH_OPTIMIZE_SPEED TRUE`
OS optimization.

Subsystem options

- `#define CH_USE_REGISTRY TRUE`
Threads registry APIs.
- `#define CH_USE_WAITEXIT TRUE`
Threads synchronization APIs.
- `#define CH_USE_SEMAPHORES TRUE`
Semaphores APIs.
- `#define CH_USE_SEMAPHORES_PRIORITY FALSE`
Semaphores queuing mode.
- `#define CH_USE_SEMSW TRUE`
Atomic semaphore API.
- `#define CH_USE_MUTEXES TRUE`
Mutexes APIs.
- `#define CH_USE_CONDVARS TRUE`
Conditional Variables APIs.
- `#define CH_USE_CONDVARS_TIMEOUT TRUE`
Conditional Variables APIs with timeout.
- `#define CH_USE_EVENTS TRUE`
Events Flags APIs.
- `#define CH_USE_EVENTS_TIMEOUT TRUE`
Events Flags APIs with timeout.
- `#define CH_USE_MESSAGES TRUE`
Synchronous Messages APIs.
- `#define CH_USE_MESSAGES_PRIORITY FALSE`
Synchronous Messages queuing mode.
- `#define CH_USE_MAILBOXES TRUE`
Mailboxes APIs.
- `#define CH_USE_QUEUES TRUE`
I/O Queues APIs.
- `#define CH_USE_MEMCORE TRUE`
Core Memory Manager APIs.
- `#define CH_USE_HEAP TRUE`
Heap Allocator APIs.
- `#define CH_USE_MALLOC_HEAP FALSE`
C-runtime allocator.
- `#define CH_USE_MEMPOOLS TRUE`
Memory Pools Allocator APIs.
- `#define CH_USE_DYNAMIC TRUE`
Dynamic Threads APIs.

Debug options

- `#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 TRUE`
Debug option, threads profiling.

Kernel hooks

- `#define THREAD_EXT_FIELDS`
Threads descriptor structure extension.
- `#define THREAD_EXT_INIT_HOOK(tp)`
Threads initialization hook.
- `#define THREAD_EXT_EXIT_HOOK(tp)`
Threads finalization hook.
- `#define THREAD_CONTEXT_SWITCH_HOOK(ntp, otp)`
Context switch hook.
- `#define IDLE_LOOP_HOOK()`
Idle Loop hook.
- `#define SYSTEM_TICK_EVENT_HOOK()`
System tick event hook.
- `#define SYSTEM_HALT_HOOK()`
System halt hook.

10.3.2 Define Documentation

10.3.2.1 `#define CH_FREQUENCY 1000`

System tick frequency.

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

10.3.2.2 `#define CH_TIME_QUANTUM 20`

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.

10.3.2.3 #define CH_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_USE_MEMCORE`.

10.3.2.4 #define CH_NO_IDLE_THREAD FALSE

Idle thread automatic spawn suppression.

When this option is activated the function `chSysInit()` does not spawn the idle thread automatically. The application has then the responsibility to do one of the following:

- Spawn a custom idle thread at priority `IDLEPRIO`.
- Change the `main()` thread priority to `IDLEPRIO` then enter an endless loop. In this scenario the `main()` thread acts as the idle thread.

Note

Unless an idle thread is spawned the `main()` thread must not enter a sleep state.

10.3.2.5 #define CH_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`.

10.3.2.6 #define CH_USE_REGISTRY TRUE

Threads registry APIs.

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

Note

The default is `TRUE`.

10.3.2.7 #define CH_USE_WAITEXIT TRUE

Threads synchronization APIs.

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

Note

The default is `TRUE`.

10.3.2.8 #define CH_USE_SEMAPHORES TRUE

Semaphores APIs.

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

Note

The default is TRUE.

10.3.2.9 #define CH_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_USE_SEMAPHORES.

10.3.2.10 #define CH_USE_SEMSW TRUE

Atomic semaphore API.

If enabled then the semaphores the [chSemSignalWait\(\)](#) API is included in the kernel.

Note

The default is TRUE.

Requires CH_USE_SEMAPHORES.

10.3.2.11 #define CH_USE_MUTEXES TRUE

Mutexes APIs.

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

Note

The default is TRUE.

10.3.2.12 #define CH_USE_CONDVAR TRUE

Conditional Variables APIs.

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

Note

The default is TRUE.

Requires CH_USE_MUTEXES.

10.3.2.13 #define CH_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_USE_CONDVAR.

10.3.2.14 #define CH_USE_EVENTS TRUE

Events Flags APIs.

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

Note

The default is TRUE.

10.3.2.15 #define CH_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_USE_EVENTS.

10.3.2.16 #define CH_USE_MESSAGES TRUE

Synchronous Messages APIs.

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

Note

The default is TRUE.

10.3.2.17 #define CH_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_USE_MESSAGES.

10.3.2.18 #define CH_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_USE_SEMAPHORES.

10.3.2.19 #define CH_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.

10.3.2.20 #define CH_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.

10.3.2.21 #define CH_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_USE_MEMCORE and either CH_USE_MUTEXES or CH_USE_SEMAPHORES.

Mutexes are recommended.

10.3.2.22 #define CH_USE_MALLOC_HEAP FALSE

C-runtime allocator.

If enabled the the heap allocator APIs just wrap the C-runtime `malloc()` and `free()` functions.

Note

The default is FALSE.

Requires CH_USE_HEAP.

The C-runtime may or may not require CH_USE_MEMCORE, see the appropriate documentation.

10.3.2.23 #define CH_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.

10.3.2.24 #define CH_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_USE_WAITEXIT.

Requires CH_USE_HEAP and/or CH_USE_MEMPOOLS.

10.3.2.25 #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.

10.3.2.26 #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.

10.3.2.27 #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.

10.3.2.28 #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.

10.3.2.29 #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.

10.3.2.30 #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.

10.3.2.31 #define CH_DBG_THREADS_PROFILING TRUE

Debug option, threads profiling.

If enabled then a field is added to the [Thread](#) structure that counts the system ticks occurred while executing the thread.

Note

The default is TRUE.

This debug option is defaulted to TRUE because it is required by some test cases into the test suite.

10.3.2.32 #define THREAD_EXT_FIELDS

Threads descriptor structure extension.

User fields added to the end of the [Thread](#) structure.

10.3.2.33 #define THREAD_EXT_INIT_HOOK(tp)**Value:**

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

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.

10.3.2.34 `#define 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.

10.3.2.35 `#define THREAD_CONTEXT_SWITCH_HOOK(ntp, otp)`

Value:

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

Context switch hook.

This hook is invoked just before switching between threads.

10.3.2.36 `#define IDLE_LOOP_HOOK()`

Value:

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

Idle Loop hook.

This hook is continuously invoked by the idle thread loop.

10.3.2.37 `#define SYSTEM_TICK_EVENT_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.

10.3.2.38 #define SYSTEM_HALT_HOOK()

Value:

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

System halt hook.

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

10.4 Types

10.4.1 Detailed Description

The system types are defined into the port layer, please refer to the core port implementation section.

System types and macros.

Defines

- #define **INLINE** inline
Inline function modifier.
- #define **ROMCONST** const
ROM constant modifier.
- #define **PACK_STRUCT_STRUCT** __attribute__((packed))
Packed structure modifier (within).
- #define **PACK_STRUCT_BEGIN**
Packed structure modifier (before).
- #define **PACK_STRUCT_END**
Packed structure modifier (after).

Typedefs

- typedef int32_t **bool_t**
Boolean, recommended the fastest signed.
- typedef uint8_t **tmode_t**
Thread mode flags, uint8_t is ok.
- typedef uint8_t **tstate_t**
Thread state, uint8_t is ok.
- typedef uint8_t **trefs_t**
Thread references counter, uint8_t is ok.
- typedef uint32_t **tprio_t**
Priority, use the fastest unsigned type.
- typedef int32_t **msg_t**
Message, use signed pointer equivalent.
- typedef int32_t **eventid_t**
Event Id, use fastest signed.
- typedef uint32_t **eventmask_t**
Event Mask, recommended fastest unsigned.
- typedef uint32_t **systime_t**
System Time, recommended fastest unsigned.
- typedef int32_t **cnt_t**
Counter, recommended fastest signed.

10.4.2 Define Documentation

10.4.2.1 #define INLINE inline

Inline function modifier.

10.4.2.2 #define ROMCONST const

ROM constant modifier.

Note

This is required because some compilers require a custom keyword, usually this macro is just set to "const" for the GCC compiler.

This macro is not used to place constants in different address spaces (like AVR requires for example) because it is assumed that a pointer to a ROMCONST constant is compatible with a pointer to a normal variable. It is just like the "const" keyword but requires that the constant is placed in ROM if the architecture supports it.

10.4.2.3 #define PACK_STRUCT_STRUCT __attribute__((packed))

Packed structure modifier (within).

10.4.2.4 #define PACK_STRUCT_BEGIN

Packed structure modifier (before).

10.4.2.5 #define PACK_STRUCT_END

Packed structure modifier (after).

10.4.3 Typedef Documentation

10.4.3.1 typedef int32_t bool_t

Boolean, recommended the fastest signed.

10.4.3.2 typedef uint8_t tmode_t

[Thread](#) mode flags, uint8_t is ok.

10.4.3.3 typedef uint8_t tstate_t

[Thread](#) state, uint8_t is ok.

10.4.3.4 typedef uint8_t trefs_t

[Thread](#) references counter, uint8_t is ok.

10.4.3.5 typedef uint32_t tprio_t

Priority, use the fastest unsigned type.

10.4.3.6 `typedef int32_t msg_t`

Message, use signed pointer equivalent.

10.4.3.7 `typedef int32_t eventid_t`

Event Id, use fastest signed.

10.4.3.8 `typedef uint32_t eventmask_t`

Event Mask, recommended fastest unsigned.

10.4.3.9 `typedef uint32_t systime_t`

System Time, recommended fastest unsigned.

10.4.3.10 `typedef int32_t cnt_t`

Counter, recommended fastest signed.

10.5 Base Kernel Services

10.5.1 Detailed Description

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

Modules

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

10.6 System Management

10.6.1 Detailed Description

System related APIs and services:

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

Functions

- void `chSysInit` (void)
ChibiOS/RT initialization.
- void `chSysTimerHandler1` (void)
Handles time ticks for round robin preemption and timer increments.
- `WORKING_AREA` (`_idle_thread_wa`, `PORT_IDLE_THREAD_STACK_SIZE`)
Idle thread working area.
- void `_idle_thread` (void *p)
This function implements the idle thread infinite loop.

Macro Functions

- #define `chSysGetIdleThread()` (`rlist.r_queue.p_prev`)
Returns a pointer to the idle thread.
- #define `chSysHalt()` `port_halt()`
Halts the system.
- #define `chSysSwitch`(ntp, otp)
Performs a context switch.
- #define `chSysDisable()`
Raises the system interrupt priority mask to the maximum level.
- #define `chSysSuspend()`
Raises the system interrupt priority mask to system level.
- #define `chSysEnable()`
Lowers the system interrupt priority mask to user level.
- #define `chSysLock()`
Enters the kernel lock mode.
- #define `chSysUnlock()`
Leaves the kernel lock mode.
- #define `chSysLockFromIsr()`
Enters the kernel lock mode from within an interrupt handler.
- #define `chSysUnlockFromIsr()`
Leaves the kernel lock mode from within an interrupt handler.

ISRs abstraction macros

- #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.

10.6.2 Function Documentation

10.6.2.1 void chSysInit(void)

ChibiOS/RT initialization.

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

Precondition

Interrupts must be still disabled when `chSysInit()` is invoked and are internally enabled.

Postcondition

The main thread is created with priority NORMALPRIO.

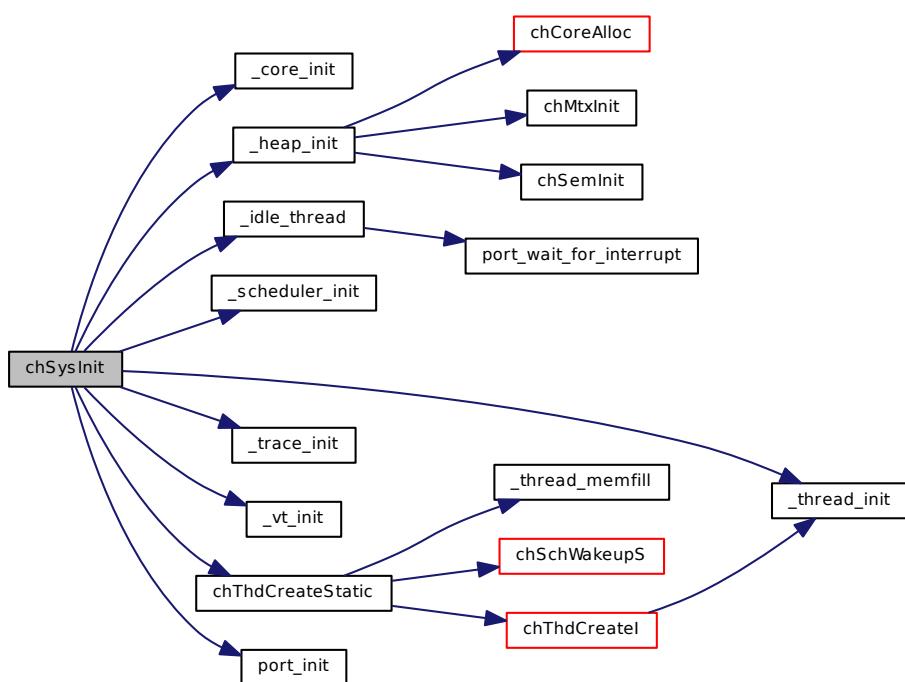
Note

This function has special, architecture-dependent, requirements, see the notes into the various port reference manuals.

Function Class:

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

Here is the call graph for this function:



10.6.2.2 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.

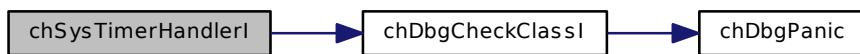
Note

The frequency of the timer determines the system tick granularity and, together with the CH_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:

**10.6.2.3 WORKING_AREA (_idle_thread_wa , PORT_IDLE_THREAD_STACK_SIZE)**

Idle thread working area.

10.6.2.4 void _idle_thread (void * p)

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:

**10.6.3 Define Documentation****10.6.3.1 #define chSysGetIdleThread() (rlist.r_queue.p_prev)**

Returns a pointer to the idle thread.

Precondition

In order to use this function the option CH_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:

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

10.6.3.2 #define chSysHalt() port_halt()

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.

Function Class:

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

10.6.3.3 #define chSysSwitch(ntp, otp)**Value:**

```
{                                     \
    dbg_trace(otp);                   \
    THREAD_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.

10.6.3.4 #define chSysDisable()**Value:**

```
{
    port_disable();
    \dbg_check_disable();
}
```

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.

10.6.3.5 #define chSysSuspend()

Value:

```
{
    port_suspend();
    \dbg_check_suspend();
}
```

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.

10.6.3.6 #define chSysEnable()

Value:

```
{
    \dbg_check_enable();
    port_enable();
}
```

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.

10.6.3.7 #define chSysLock()

Value:

```
{
    port_lock();
    dbg_check_lock();
}
```

Enters the kernel lock mode.

Function Class:

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

10.6.3.8 #define chSysUnlock()

Value:

```
{
    dbg_check_unlock();
    port_unlock();
}
```

Leaves the kernel lock mode.

Function Class:

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

10.6.3.9 #define chSysLockFromIsr()

Value:

```
{
    port_lock_from_isr();
    dbg_check_lock_from_isr();
}
```

Enters the kernel lock mode 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.

10.6.3.10 #define chSysUnlockFromIsr()

Value:

```
{
    dbg_check_unlock_from_isr();
    port_unlock_from_isr();
}
```

Leaves the kernel lock mode 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.

10.6.3.11 #define CH_IRQ_PROLOGUE()**Value:**

```
PORT_IRQ_PROLOGUE();  
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.

10.6.3.12 #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.

10.6.3.13 #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.

```
10.6.3.14 #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.

10.7 Scheduler

10.7.1 Detailed Description

This module provides the default portable scheduler code, scheduler functions can be individually captured by the port layer in order to provide architecture optimized equivalents. When a function is captured its default code is not built into the OS image, the optimized version is included instead.

Data Structures

- struct [ReadyList](#)

Ready list header.

Functions

- void [_scheduler_init](#) (void)
Scheduler initialization.
- [Thread * chSchReadyI](#) ([Thread](#) *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](#) *ntp, [msg_t](#) msg)
Wakes up a thread.
- void [chSchRescheduleS](#) (void)
Performs a reschedule if a higher priority thread is runnable.
- [bool_t 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

- [ReadyList rlist](#)

Ready list header.

Wakeup status codes

- #define RDY_OK 0
Normal wakeup message.
- #define RDY_TIMEOUT -1
Wakeup caused by a timeout condition.
- #define RDY_RESET -2
Wakeup caused by a reset condition.

Priority constants

- #define NOPRIO 0
Ready list header priority.
- #define IDLEPRIO 1
Idle thread priority.
- #define LOWPRIO 2
Lowest user priority.
- #define NORMALPRIO 64
Normal user priority.
- #define HIGHPRIO 127
Highest user priority.
- #define ABSPRIO 255
Greatest possible priority.

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.

Macro Functions

- #define chSchIsRescRequired() (firstprio(&rlist.r_queue) > currp->p_prio)
Determines if the current thread must reschedule.
- #define chSchCanYieldS() (firstprio(&rlist.r_queue) >= currp->p_prio)
Determines if yielding is possible.
- #define chSchDoYieldS()
Yields the time slot.
- #define chSchPreemption()
Inline-able preemption code.

Defines

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

10.7.2 Function Documentation

10.7.2.1 void _scheduler_init(void)

Scheduler initialization.

Function Class:

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

10.7.2.2 Thread * chSchReadyI(Thread * 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 `tp` the thread to be made ready

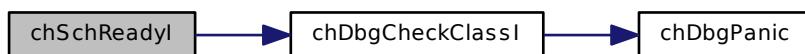
Returns

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:



10.7.2.3 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`.

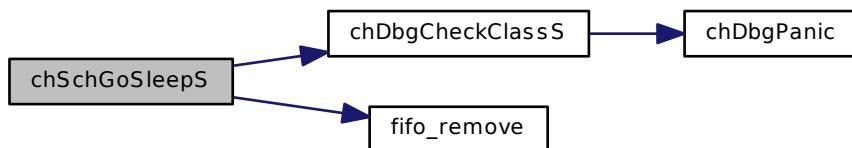
Parameters

in *newstate* 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:



10.7.2.4 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 `RDY_TIMEOUT` low level message. The possible [Thread States](#) are defined into `threads.h`.

Parameters

in *newstate* the new thread state

in *time* the number of ticks before the operation timeouts, the special values are handled as follow:

- `TIME_INFINITE` the thread enters an infinite sleep state, this is equivalent to invoking [chSchGoSleepS\(\)](#) but, of course, less efficient.
- `TIME_IMMEDIATE` this value is not allowed.

Returns

The wakeup message.

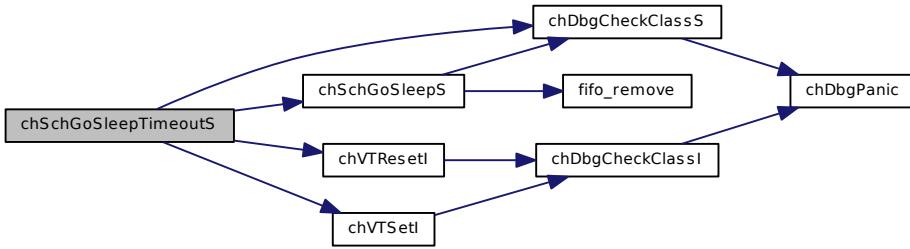
Return values

`RDY_TIMEOUT` 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:



10.7.2.5 void chSchWakeupS (Thread * *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.

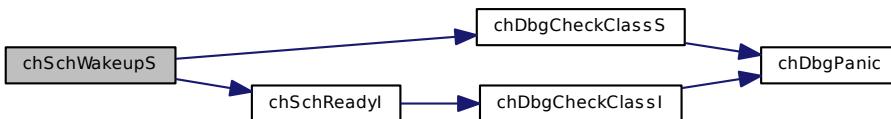
Parameters

in	<i>ntp</i>	the Thread to be made ready
in	<i>msg</i>	message to the awakened thread

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:



10.7.2.6 void chSchRescheduleS (void)

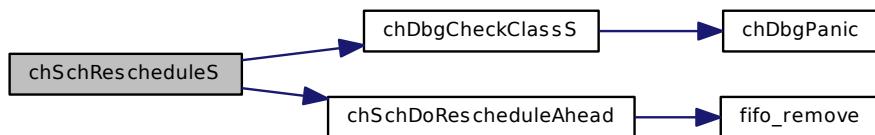
Performs a reschedule if a higher priority thread is runnable.

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:



10.7.2.7 `bool_t 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.

10.7.2.8 `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.

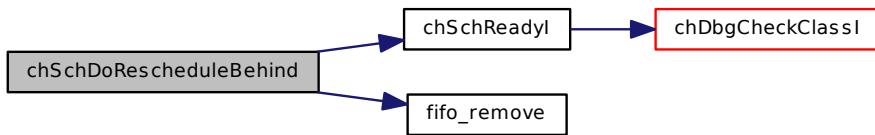
Note

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:



10.7.2.9 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:



10.7.2.10 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.

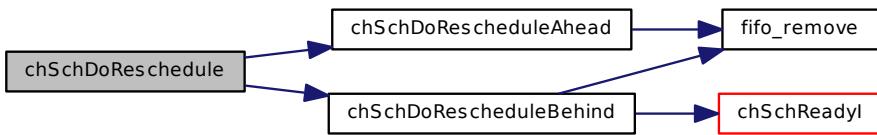
Note

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:



10.7.3 Variable Documentation

10.7.3.1 ReadyList rlist

Ready list header.

10.7.4 Define Documentation

10.7.4.1 #define RDY_OK 0

Normal wakeup message.

10.7.4.2 #define RDY_TIMEOUT -1

Wakeup caused by a timeout condition.

10.7.4.3 #define RDY_RESET -2

Wakeup caused by a reset condition.

10.7.4.4 #define NOPRIO 0

Ready list header priority.

10.7.4.5 #define IDLEPRIO 1

Idle thread priority.

10.7.4.6 #define LOWPRIO 2

Lowest user priority.

10.7.4.7 #define NORMALPRIO 64

Normal user priority.

10.7.4.8 #define HIGHPRIO 127

Highest user priority.

10.7.4.9 #define ABSPRIO 255

Greatest possible priority.

10.7.4.10 #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.

10.7.4.11 #define TIME_INFINITE ((systime_t)-1)

Infinite time specification for all functions with a timeout specification.

10.7.4.12 #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.

10.7.4.13 #define currp rlist.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.

10.7.4.14 #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.

10.7.4.15 #define chSchIsRescRequired() (firstprio(&rlist.r_queue) > currp->p_prio)

Determines if the current thread must reschedule.

This function returns TRUE if there is a ready thread with 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.

10.7.4.16 #define chSchCanYieldS() (firstprio(&rlist.r_queue) >= currp->p_prio)

Determines if yielding is possible.

This function returns TRUE if there is a ready thread with 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.

10.7.4.17 #define chSchDoYieldS()

Value:

```
{
    if (chSchCanYieldS())
        chSchDoRescheduleBehind();
}
```

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.

10.7.4.18 #define chSchPreemption()

Value:

```
{
    tprio_t p1 = firstprio(&rlist.r_queue);
    tprio_t p2 = currp->p_prio;
    if (currp->p_preempt) {
        if (p1 > p2)
            chSchDoRescheduleAhead();
    }
    else {
        if (p1 >= p2)
            chSchDoRescheduleBehind();
    }
}
```

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.

10.8 Threads

10.8.1 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_USE_WAITEXIT` and `CH_USE_DYNAMIC` configuration options.

Data Structures

- struct `Thread`
Structure representing a thread.

Functions

- `Thread * _thread_init (Thread *tp, tprio_t prio)`
Initializes a thread structure.
- `void _thread_memfill (uint8_t *startp, uint8_t *endp, uint8_t v)`
Memory fill utility.
- `Thread * chThdCreate (void *wsp, size_t size, tprio_t prio, tfunc_t pf, void *arg)`
Creates a new thread into a static memory area.
- `Thread * chThdCreateStatic (void *wsp, size_t size, tprio_t prio, tfunc_t pf, void *arg)`
Creates a new thread into a static memory area.
- `tprio_t chThdSetPriority (tprio_t newprio)`
Changes the running thread priority level then reschedules if necessary.
- `Thread * chThdResume (Thread *tp)`
Resumes a suspended thread.
- `void chThdTerminate (Thread *tp)`
Requests a thread termination.
- `void chThdSleep (systime_t time)`

- `void chThdSleepUntil (systime_t time)`
Suspends the invoking thread for the specified time.
- `void chThdYield (void)`
Suspends the invoking thread until the system time arrives to the specified value.
- `void chThdExit (msg_t msg)`
Yields the time slot.
- `void chThdExitS (msg_t msg)`
Terminates the current thread.
- `msg_t chThdWait (Thread *tp)`
Terminates the current thread.
- `msg_t chThdWait (Thread *tp)`
Blocks the execution of the invoking thread until the specified thread terminates then the exit code is returned.

Thread states

- `#define THD_STATE_READY 0`
Waiting on the ready list.
- `#define THD_STATE_CURRENT 1`
Currently running.
- `#define THD_STATE_SUSPENDED 2`
Created in suspended state.
- `#define THD_STATE_WTSEM 3`
Waiting on a semaphore.
- `#define THD_STATE_WTMutex 4`
Waiting on a mutex.
- `#define THD_STATE_WTCOND 5`
Waiting on a condition variable.
- `#define THD_STATE_SLEEPING 6`
Waiting in `chThdSleep ()` or `chThdSleepUntil ()`.
- `#define THD_STATE_WTEXIT 7`
Waiting in `chThdWait ()`.
- `#define THD_STATE_WTOREVT 8`
Waiting for an event.
- `#define THD_STATE_WTANDEVT 9`
Waiting for several events.
- `#define THD_STATE SNDMSGQ 10`
Sending a message, in queue.
- `#define THD_STATE SNDMSG 11`
Sent a message, waiting answer.
- `#define THD_STATE_WTMSG 12`
Waiting for a message.
- `#define THD_STATE_WTQUEUE 13`
Waiting on an I/O queue.
- `#define THD_STATE_FINAL 14`
Thread terminated.
- `#define THD_STATE_NAMES`
Thread states as array of strings.

Thread flags and attributes

- #define THD_MEM_MODE_MASK 3
Thread memory mode mask.
- #define THD_MEM_MODE_STATIC 0
Static thread.
- #define THD_MEM_MODE_HEAP 1
Thread allocated from a Memory Heap.
- #define THD_MEM_MODE_MEMPOOL 2
Thread allocated from a Memory Pool.
- #define THD_TERMINATE 4
Termination requested flag.

Macro Functions

- #define chThdSelf() currp
Returns a pointer to the current Thread.
- #define chThdGetPriority() (currp->p_prio)
Returns the current thread priority.
- #define chThdGetTicks(tp) ((tp)->p_time)
Returns the number of ticks consumed by the specified thread.
- #define chThdLS() (void *) (currp + 1)
Returns the pointer to the Thread local storage area, if any.
- #define chThdTerminated(tp) ((tp)->p_state == THD_STATE_FINAL)
Verifies if the specified thread is in the THD_STATE_FINAL state.
- #define chThdShouldTerminate() (currp->p_flags & THD_TERMINATE)
Verifies if the current thread has a termination request pending.
- #define chThdResumel(tp) chSchReadyI(tp)
Resumes a thread created with chThdCreateI().
- #define chThdSleepS(time) chSchGoSleepTimeoutS(THD_STATE_SLEEPING, time)
Suspends the invoking thread for the specified time.
- #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 msg_t(* tfunc_t)(void *)
Thread function.

10.8.2 Function Documentation

10.8.2.1 Thread *_thread_init (Thread * 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.

10.8.2.2 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.

10.8.2.3 Thread * chThdCreate (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 THD_STATE_SUSPENDED.

Postcondition

The initialized thread can be subsequently started by invoking [chThdResume \(\)](#), [chThdResumeI \(\)](#) 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.

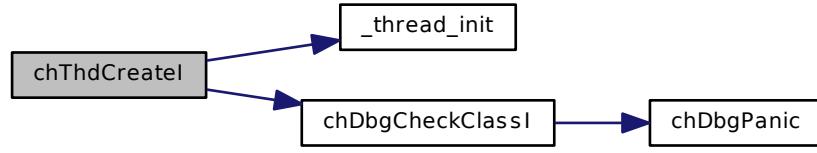
Returns

The pointer to the [Thread](#) 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:



10.8.2.4 Thread * 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	<code>wsp</code>	pointer to a working area dedicated to the thread stack
in	<code>size</code>	size of the working area
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.

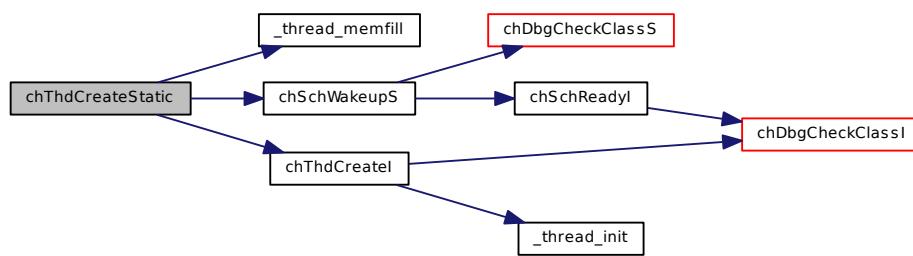
Returns

The pointer to the `Thread` 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:



10.8.2.5 `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 *newprio* the new priority level of the running thread

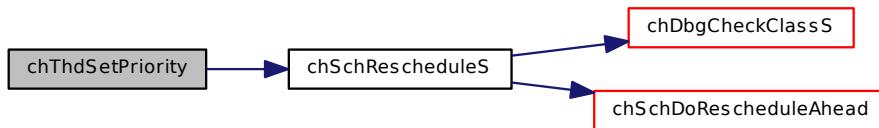
Returns

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:



10.8.2.6 `Thread * chThdResume (Thread * tp)`

Resumes a suspended thread.

Precondition

The specified thread pointer must refer to an initialized thread in the THD_STATE_SUSPENDED state.

Postcondition

The specified thread is immediately started or put in the ready list depending on the relative priority levels.

Note

Use this function to start threads created with [chThdCreateI\(\)](#).

Parameters

in *tp* pointer to the thread

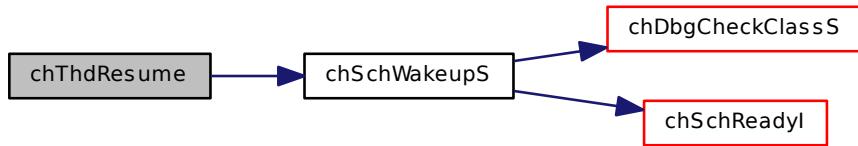
Returns

The 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:



10.8.2.7 void chThdTerminate (Thread * 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 *tp* pointer to the thread

Function Class:

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

10.8.2.8 void chThdSleep (systime_t time)

Suspends the invoking thread for the specified time.

Parameters

in *time* the delay in system ticks, the special values are handled as follow:

- *TIME_INFINITE* the thread enters an infinite sleep state.
- *TIME_IMMEDIATE* 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.

10.8.2.9 void chThdSleepUntil (systime_t time)

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

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.

10.8.2.10 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.

10.8.2.11 void chThdExit (msg_t msg)

Terminates the current thread.

The thread goes in the THD_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.

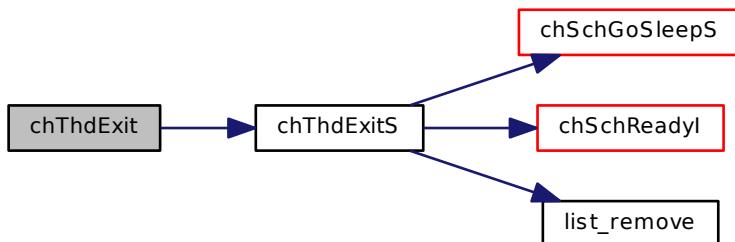
Parameters

in msg thread exit code

Function Class:

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:

**10.8.2.12 void chThdExitS (msg_t msg)**

Terminates the current thread.

The thread goes in the THD_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.

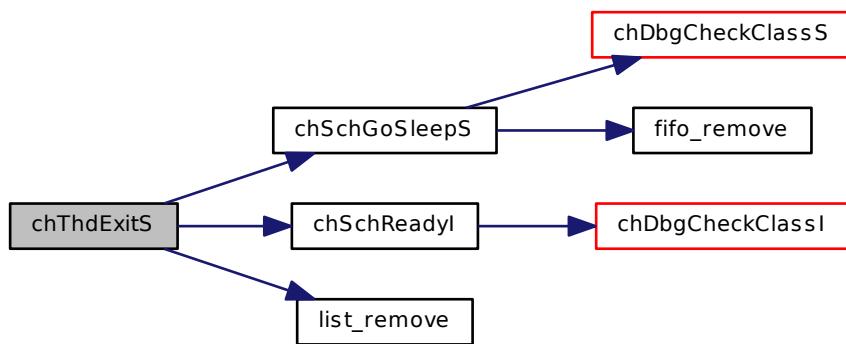
Parameters

in	<i>msg</i> 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:

**10.8.2.13 msg_t chThdWait(Thread * 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_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` structure. After invoking `chThdWait()` the thread pointer becomes invalid and must not be used as parameter for further system calls.

Note

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

Parameters

in *tp* pointer to the thread

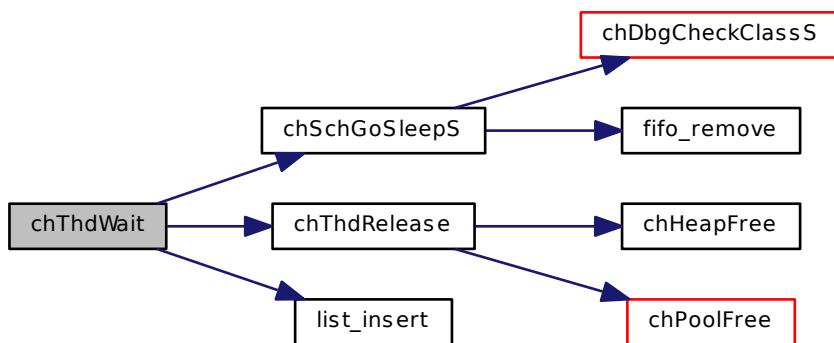
Returns

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:

**10.8.3 Define Documentation****10.8.3.1 #define THD_STATE_READY 0**

Waiting on the ready list.

10.8.3.2 #define THD_STATE_CURRENT 1

Currently running.

10.8.3.3 #define THD_STATE_SUSPENDED 2

Created in suspended state.

10.8.3.4 #define THD_STATE_WTSEM 3

Waiting on a semaphore.

10.8.3.5 #define THD_STATE_WTMTX 4

Waiting on a mutex.

10.8.3.6 #define THD_STATE_WTCOND 5

Waiting on a condition variable.

10.8.3.7 #define THD_STATE_SLEEPING 6

Waiting in `chThdSleep()` or `chThdSleepUntil()`.

10.8.3.8 #define THD_STATE_WTEXIT 7

Waiting in `chThdWait()`.

10.8.3.9 #define THD_STATE_WTOREVT 8

Waiting for an event.

10.8.3.10 #define THD_STATE_WTANDEVT 9

Waiting for several events.

10.8.3.11 #define THD_STATE SNDMSGQ 10

Sending a message, in queue.

10.8.3.12 #define THD_STATE SNDMSG 11

Sent a message, waiting answer.

10.8.3.13 #define THD_STATE_WTMSG 12

Waiting for a message.

10.8.3.14 #define THD_STATE_WTQUEUE 13

Waiting on an I/O queue.

10.8.3.15 #define THD_STATE_FINAL 14

`Thread` terminated.

10.8.3.16 #define THD_STATE_NAMES

Value:

```
"READY", "CURRENT", "SUSPENDED", "WTSEM", "WTMTX", "WTCOND", "SLEEPING", \
"WTEXIT", "WTOREVT", "WTANDEVT", "SNDMSGQ", "SNDMSG", "WTMSG", "WTQUEUE", \
"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.

10.8.3.17 `#define THD_MEM_MODE_MASK 3`

`Thread` memory mode mask.

10.8.3.18 `#define THD_MEM_MODE_STATIC 0`

Static thread.

10.8.3.19 `#define THD_MEM_MODE_HEAP 1`

`Thread` allocated from a Memory Heap.

10.8.3.20 `#define THD_MEM_MODE_MEMPOOL 2`

`Thread` allocated from a Memory Pool.

10.8.3.21 `#define THD_TERMINATE 4`

Termination requested flag.

10.8.3.22 `#define chThdSelf() currp`

Returns a pointer to the current `Thread`.

Note

Can be invoked in any context.

Function Class:

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

10.8.3.23 `#define chThdGetPriority() (currp->p.prio)`

Returns the current thread priority.

Note

Can be invoked in any context.

Function Class:

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

10.8.3.24 `#define chThdGetTicks(tp) ((tp)->p.time)`

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.
Can be invoked in any context.

Parameters

in *tp* pointer to the thread

Function Class:

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

10.8.3.25 #define chThdLS() (void *)(currp + 1)

Returns the pointer to the [Thread](#) local storage area, if any.

Note

Can be invoked in any context.

Function Class:

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

10.8.3.26 #define chThdTerminated(*tp*) ((tp)->p_state == THD_STATE_FINAL)

Verifies if the specified thread is in the THD_STATE_FINAL state.

Note

Can be invoked in any context.

Parameters

in *tp* pointer to the thread

Return values

TRUE thread terminated.

FALSE thread not terminated.

Function Class:

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

10.8.3.27 #define chThdShouldTerminate() (currp->p_flags & THD_TERMINATE)

Verifies if the current thread has a termination request pending.

Note

Can be invoked in any context.

Return values

0 termination request not pending.

/0 termination request pending.

Function Class:

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

10.8.3.28 #define chThdResumel(*tp*) chSchReadyI(*tp*)

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

Parameters

in *tp* pointer to the thread

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.

10.8.3.29 #define chThdSleepS(*time*) chSchGoSleepTimeoutS(THD_STATE_SLEEPING, *time*)

Suspends the invoking thread for the specified time.

Parameters

in *time* the delay in system ticks, the special values are handled as follow:

- *TIME_INFINITE* the thread enters an infinite sleep state.
- *TIME_IMMEDIATE* 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.

10.8.3.30 #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 *sec* 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.

10.8.3.31 #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 *msec* 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.

10.8.3.32 #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 *usec* 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.

10.8.4 Typedef Documentation

10.8.4.1 typedef msg_t(* tfunc_t)(void *)

Thread function.

10.9 Time and Virtual Timers**10.9.1 Detailed Description**

Time and Virtual Timers related APIs and services.

Data Structures

- struct [VirtualTimer](#)
Virtual Timer descriptor structure.
- struct [VTList](#)
Virtual timers list header.

Functions

- void [_vt_init](#) (void)
Virtual Timers initialization.
- void [chVTSel](#) ([VirtualTimer](#) *vtp, [systime_t](#) time, [vfunc_t](#) vfunc, void *par)
Enables a virtual timer.
- void [chVTRestetl](#) ([VirtualTimer](#) *vtp)
Disables a Virtual Timer.

Variables

- **VTList vlist**
Virtual timers delta list header.
- **VTList vlist**
Virtual timers delta list header.

Time conversion utilities

- **#define S2ST(sec) ((systime_t)((sec) * CH_FREQUENCY))**
Seconds to system ticks.
- **#define MS2ST(msec)**
Milliseconds to system ticks.
- **#define US2ST(usec)**
Microseconds to system ticks.

Macro Functions

- **#define chVTDoTick()**
Virtual timers ticker.
- **#define chVTIsArmed(vtp) ((vtp)->vt_func != NULL)**
Returns TRUE if the specified timer is armed.
- **#define chVTSet(vtp, time, vfunc, par)**
Enables a virtual timer.
- **#define chVTRest(vtp)**
Disables a Virtual Timer.
- **#define chTimeNow() (vlist.vt_systime)**
Current system time.
- **#define chTimeElapsedSince(start) (chTimeNow() - (start))**
Returns the elapsed time since the specified start time.
- **#define chTimelsWithin(start, end) (chTimeElapsedSince(start) < ((end) - (start)))**
Checks if the current system time is within the specified time window.

TypeDefs

- **typedef void(* vfunc_t)(void *)**
Virtual Timer callback function.
- **typedef struct VirtualTimer VirtualTimer**
Virtual Timer structure type.

10.9.2 Function Documentation

10.9.2.1 void _vt_init(void)

Virtual Timers initialization.

Note

Internal use only.

Function Class:

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

10.9.2.2 void chVTSetl (VirtualTimer * vtp, systime_t time, vtfunc_t vtfunc, void * par)

Enables a virtual timer.

Note

The associated function is invoked from interrupt context.

Parameters

out	<i>vtp</i>	the <code>VirtualTimer</code> structure pointer
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> is allowed but interpreted as a normal time specification. • <code>TIME_IMMEDIATE</code> this value is not allowed.
in	<i>vtfunc</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:



10.9.2.3 void chVTResetl (VirtualTimer * vtp)

Disables a Virtual Timer.

Note

The timer MUST be active when this function is invoked.

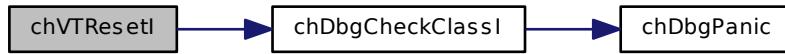
Parameters

in	<i>vtp</i>	the <code>VirtualTimer</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:



10.9.3 Variable Documentation

10.9.3.1 VTLIST vtlist

Virtual timers delta list header.

10.9.3.2 VTLIST vtlist

Virtual timers delta list header.

10.9.4 Define Documentation

10.9.4.1 #define S2ST(sec) ((systime_t)((sec) * CH_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.

10.9.4.2 #define MS2ST(msec)

Value:

```
((systime_t) (((((uint32_t)(msec)) * ((uint32_t)CH_FREQUENCY) - 1UL) /      \
1000UL) + 1UL))
```

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.

10.9.4.3 #define US2ST(*usec*)**Value:**

```
((systime_t) (((((uint32_t) (usec)) * ((uint32_t)CH_FREQUENCY) - 1UL) /      \
               1000000UL) + 1UL))
```

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.

10.9.4.4 #define chVTDoTickl()**Value:**

```
{
    vtlist.vt_systime++;                                \
    if (&vtlist != (VTLList *)vtlist.vt_next) {          \
        VirtualTimer *vtp;                             \
        \
        --vtlist.vt_next->vt_time;                     \
        while (!(vtp = vtlist.vt_next)->vt_time) {     \
            vtfunc_t fn = vtp->vt_func;                \
            vtp->vt_func = (vtfunc_t)NULL;              \
            vtp->vt_next->vt_prev = (void *)&vtlist;   \
            (&vtlist)->vt_next = vtp->vt_next;         \
            chSysUnlockFromIsr();                        \
            fn(vtp->vt_par);                          \
            chSysLockFromIsr();                         \
        }                                              \
    }                                              \
}
```

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.

10.9.4.5 #define chVTIsArmedI(*vtp*) ((*vtp*)->vt_func != NULL)

Returns TRUE if the specified 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.

10.9.4.6 #define chVTSet(*vtp*, *time*, *vfunc*, *par*)

Value:

```
{
    \           \
    chSysLock(); \
    chVTSetI(vtp, time, vfunc, par); \
    chSysUnlock(); \
}
```

Enables a virtual timer.

Note

The associated function is invoked from interrupt context.

Parameters

out	<i>vtp</i>	the VirtualTimer structure pointer
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"> • <i>TIME_INFINITE</i> is allowed but interpreted as a normal time specification. • <i>TIME_IMMEDIATE</i> 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.

10.9.4.7 #define chVTReset(*vtp*)

Value:

```
{
    \           \
    chSysLock(); \
    if (chVTIsArmedI(vtp)) \
        chVTResetI(vtp); \
    chSysUnlock(); \
}
```

Disables a Virtual Timer.

Note

The timer is first checked and disabled only if armed.

Parameters

in *vtp* the `VirtualTimer` structure pointer

Function Class:

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

10.9.4.8 #define chTimeNow() (vtlist.vt.systime)

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 is designed to work with the `chThdSleepUntil()`.

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.

10.9.4.9 #define chTimeElapsedSince(*start*) (chTimeNow() - (*start*))

Returns the elapsed time since the specified start time.

Parameters

in *start* start time

Returns

The elapsed time.

Function Class:

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

10.9.4.10 #define chTimelsWithin(*start*, *end*) (chTimeElapsedSince(*start*) < ((*end*) - (*start*)))

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.

10.9.5 Typedef Documentation

10.9.5.1 `typedef void(* vtfunc_t)(void *)`

Virtual Timer callback function.

10.9.5.2 `typedef struct VirtualTimer VirtualTimer`

Virtual Timer structure type.

10.10 Synchronization

10.10.1 Detailed Description

Synchronization services.

Modules

- [Counting Semaphores](#)
- [Binary Semaphores](#)
- [Mutexes](#)
- [Condition Variables](#)
- [Event Flags](#)
- [Synchronous Messages](#)
- [Mailboxes](#)
- [I/O Queues](#)

10.11 Counting Semaphores

10.11.1 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_USE_SEMAPHORES_PRIORITY` in `chconf.h`.

Precondition

In order to use the semaphore APIs the `CH_USE_SEMAPHORES` option must be enabled in `chconf.h`.

Data Structures

- struct `Semaphore`
`Semaphore` structure.

Functions

- void `chSemInit (Semaphore *sp, cnt_t n)`
Initializes a semaphore with the specified counter value.
- void `chSemReset (Semaphore *sp, cnt_t n)`
Performs a reset operation on the semaphore.
- void `chSemReset1 (Semaphore *sp, cnt_t n)`
Performs a reset operation on the semaphore.
- msg_t `chSemWait (Semaphore *sp)`
Performs a wait operation on a semaphore.
- msg_t `chSemWaitS (Semaphore *sp)`
Performs a wait operation on a semaphore.
- msg_t `chSemWaitTimeout (Semaphore *sp, systime_t time)`
Performs a wait operation on a semaphore with timeout specification.
- msg_t `chSemWaitTimeoutS (Semaphore *sp, systime_t time)`
Performs a wait operation on a semaphore with timeout specification.
- void `chSemSignal (Semaphore *sp)`
Performs a signal operation on a semaphore.
- void `chSemSignall (Semaphore *sp)`
Performs a signal operation on a semaphore.
- void `chSemAddCounter1 (Semaphore *sp, cnt_t n)`
Adds the specified value to the semaphore counter.
- msg_t `chSemSignalWait (Semaphore *sp, Semaphore *spw)`
Performs atomic signal and wait operations on two semaphores.

Macro Functions

- `#define chSemFastWait(sp) ((sp)->s_cnt--)`
Decreases the semaphore counter.
- `#define chSemFastSignall(sp) ((sp)->s_cnt++)`
Increases the semaphore counter.
- `#define chSemGetCounter(sp) ((sp)->s_cnt)`
Returns the semaphore counter current value.

Defines

- `#define _SEMAPHORE_DATA(name, n) {_THREADSQUEUE_DATA(name.s_queue), n}`
Data part of a static semaphore initializer.
- `#define SEMAPHORE_DECL(name, n) Semaphore name = _SEMAPHORE_DATA(name, n)`
Static semaphore initializer.

TypeDefs

- `typedef struct Semaphore Semaphore`
Semaphore structure.

10.11.2 Function Documentation

10.11.2.1 void chSemInit (**Semaphore** * *sp*, **cnt_t** *n*)

Initializes a semaphore with the specified counter value.

Parameters

out	<i>sp</i>	pointer to a Semaphore 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.

10.11.2.2 void chSemReset (**Semaphore** * *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 waked up by a reset rather than a signal because the `chSemWait()` will return `RDY_RESET` instead of `RDY_OK`.

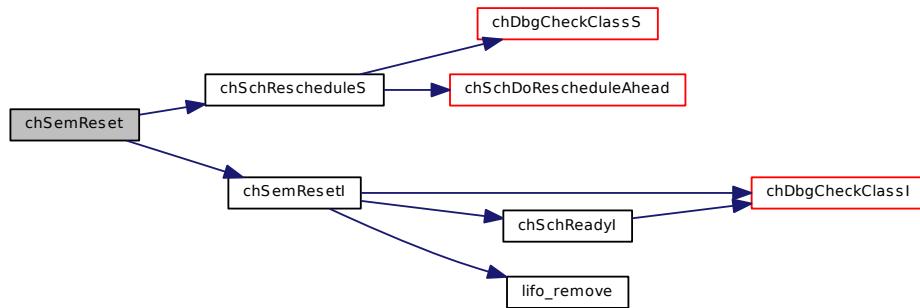
Parameters

in	<i>sp</i>	pointer to a Semaphore 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:

**10.11.2.3 void chSemResetl (Semaphore * 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 waked up by a reset rather than a signal because the `chSemWait()` will return `RDY_RESET` instead of `RDY_OK`.

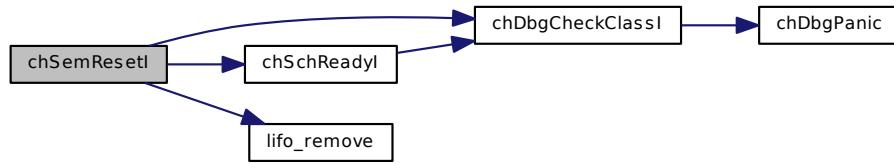
Parameters

in	<code>sp</code>	pointer to a <code>Semaphore</code> structure
in	<code>n</code>	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:



10.11.2.4 msg_t chSemWait (Semaphore * sp)

Performs a wait operation on a semaphore.

Parameters

in `sp` pointer to a [Semaphore](#) structure

Returns

A message specifying how the invoking thread has been released from the semaphore.

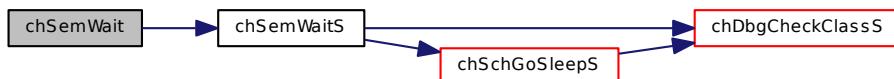
Return values

`RDY_OK` if the thread has not stopped on the semaphore or the semaphore has been signaled.
`RDY_RESET` if the semaphore has been reset using [chSemReset \(\)](#).

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:



10.11.2.5 msg_t chSemWaitS (Semaphore * sp)

Performs a wait operation on a semaphore.

Parameters

in `sp` pointer to a [Semaphore](#) structure

Returns

A message specifying how the invoking thread has been released from the semaphore.

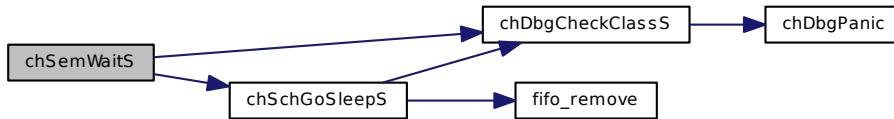
Return values

RDY_OK if the thread has not stopped on the semaphore or the semaphore has been signaled.
RDY_RESET 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:

**10.11.2.6 msg_t chSemWaitTimeout (Semaphore * sp, systime_t time)**

Performs a wait operation on a semaphore with timeout specification.

Parameters

in	<i>sp</i>	pointer to a <code>Semaphore</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"> • <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

RDY_OK if the thread has not stopped on the semaphore or the semaphore has been signaled.
RDY_RESET if the semaphore has been reset using `chSemReset ()`.
RDY_TIMEOUT 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:



10.11.2.7 `msg_t chSemWaitTimeoutS(Semaphore * sp, systime_t time)`

Performs a wait operation on a semaphore with timeout specification.

Parameters

in	<i>sp</i>	pointer to a Semaphore 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.

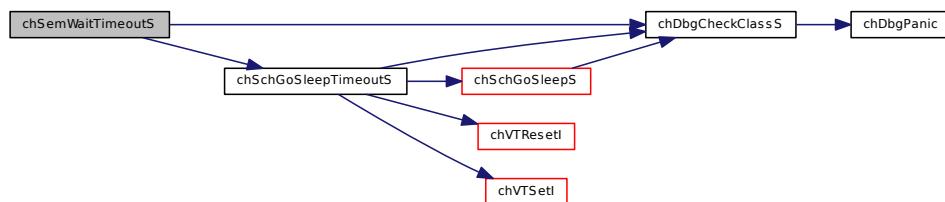
Return values

<i>RDY_OK</i>	if the thread has not stopped on the semaphore or the semaphore has been signaled.
<i>RDY_RESET</i>	if the semaphore has been reset using chSemReset() .
<i>RDY_TIMEOUT</i>	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:



10.11.2.8 `void chSemSignal(Semaphore * sp)`

Performs a signal operation on a semaphore.

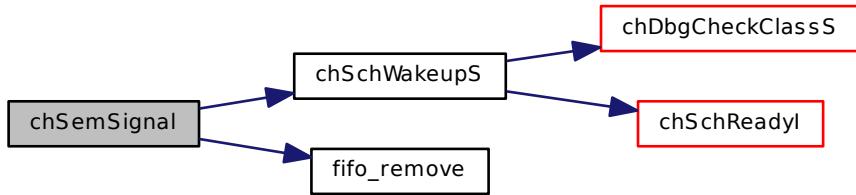
Parameters

in	<i>sp</i>	pointer to a Semaphore 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:



10.11.2.9 void chSemSignall (Semaphore * 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.

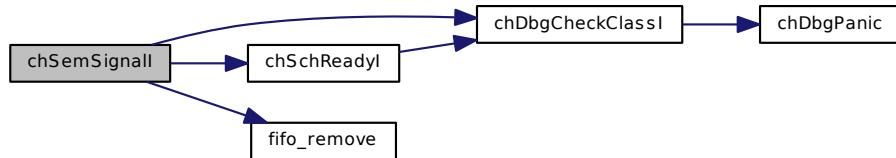
Parameters

in *sp* pointer to a `Semaphore` 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:



10.11.2.10 void chSemAddCounterl (Semaphore * 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.

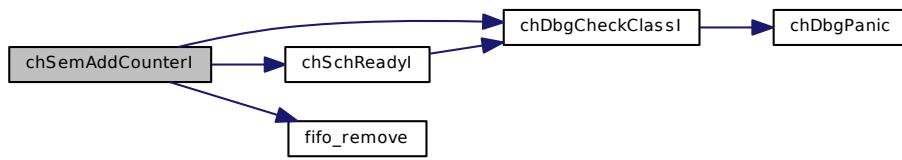
Parameters

in	<i>sp</i>	pointer to a Semaphore structure
in	<i>n</i>	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:



10.11.2.11 msg_t chSemSignalWait (Semaphore * sps, Semaphore * spw)

Performs atomic signal and wait operations on two semaphores.

Precondition

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

Parameters

in	<i>sps</i>	pointer to a Semaphore structure to be signaled
in	<i>spw</i>	pointer to a Semaphore structure to wait on

Returns

A message specifying how the invoking thread has been released from the semaphore.

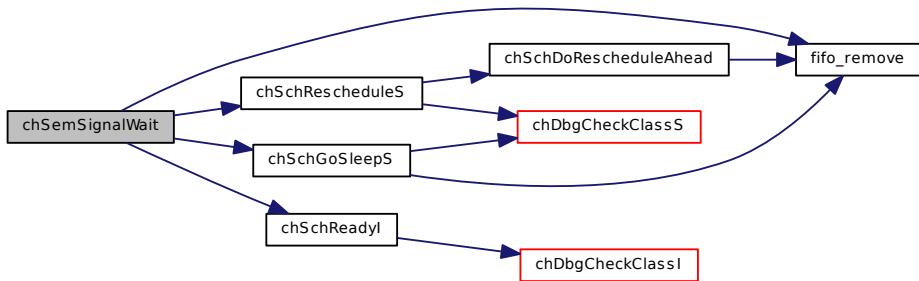
Return values

<i>RDY_OK</i>	if the thread has not stopped on the semaphore or the semaphore has been signaled.
<i>RDY_RESET</i>	if the semaphore has been reset using chSemReset () .

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:



10.11.3 Define Documentation

10.11.3.1 `#define _SEMAPHORE_DATA(name, n) {_THREADSQUEUE_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	<i>name</i>	the name of the semaphore variable
in	<i>n</i>	the counter initial value, this value must be non-negative

10.11.3.2 `#define SEMAPHORE_DECL(name, n) Semaphore 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

10.11.3.3 `#define chSemFastWaitI(sp) ((sp)->s.cnt--)`

Decreases the semaphore counter.

This macro can be used when the counter is known to 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.

10.11.3.4 `#define chSemFastSignall(sp) ((sp)->s.cnt++)`

Increases the semaphore counter.

This macro can be used when the counter is known to be not 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.

10.11.3.5 #define chSemGetCounter(sp) ((sp)->s_cnt)

Returns the semaphore counter current 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.

10.11.4 Typedef Documentation

10.11.4.1 **typedef struct Semaphore Semaphore**

[Semaphore](#) structure.

10.12 Binary Semaphores

10.12.1 Detailed Description

Binary semaphores related APIs and services.

Operation mode

Binary semaphores are implemented as a set of macros 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 the 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_USE_SEMAPHORES option must be enabled in [chconf.h](#).

Data Structures

- **struct BinarySemaphore**
Binary semaphore type.

Macro Functions

- `#define chBSemInit(bsp, taken) chSemInit(&(bsp)->bs_sem, (taken) ? 0 : 1)`
Initializes a binary semaphore.
- `#define chBSemWait(bsp) chSemWait(&(bsp)->bs_sem)`
Wait operation on the binary semaphore.
- `#define chBSemWaitS(bsp) chSemWaitS(&(bsp)->bs_sem)`
Wait operation on the binary semaphore.
- `#define chBSemWaitTimeout(bsp, time) chSemWaitTimeout(&(bsp)->bs_sem, (time))`
Wait operation on the binary semaphore.
- `#define chBSemWaitTimeoutS(bsp, time) chSemWaitTimeoutS(&(bsp)->bs_sem, (time))`
Wait operation on the binary semaphore.
- `#define chBSemReset(bsp, taken) chSemReset(&(bsp)->bs_sem, (taken) ? 0 : 1)`
Reset operation on the binary semaphore.
- `#define chBSemResetI(bsp, taken) chSemResetI(&(bsp)->bs_sem, (taken) ? 0 : 1)`
Reset operation on the binary semaphore.
- `#define chBSemSignal(bsp)`
Performs a signal operation on a binary semaphore.
- `#define chBSemSignall(bsp)`
Performs a signal operation on a binary semaphore.
- `#define chBSemGetStateI(bsp) ((bsp)->bs_sem.s_cnt > 0 ? FALSE : TRUE)`
Returns the binary semaphore current state.

Defines

- `#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) BinarySemaphore name = _BSEMAPHORE_DATA(name, taken)`
Static semaphore initializer.

10.12.2 Define Documentation

10.12.2.1 `#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	<code>name</code>	the name of the semaphore variable
in	<code>taken</code>	the semaphore initial state

10.12.2.2 `#define BSEMAPHORE_DECL(name, taken) BinarySemaphore name = _BSEMAPHORE_DATA(name, taken)`

Static semaphore initializer.

Statically initialized semaphores require no explicit initialization using `chBSemInit()`.

Parameters

in	<code>name</code>	the name of the semaphore variable
in	<code>taken</code>	the semaphore initial state

10.12.2.3 #define chBSemInit(*bsp*, *taken*) chSemInit(&(*bsp*)>bs_sem, (*taken*) ? 0 : 1)

Initializes a binary semaphore.

Parameters

out	<i>bsp</i>	pointer to a BinarySemaphore 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.

10.12.2.4 #define chBSemWait(*bsp*) chSemWait(&(*bsp*)>bs_sem)

Wait operation on the binary semaphore.

Parameters

in	<i>bsp</i>	pointer to a BinarySemaphore structure
----	------------	--

Returns

A message specifying how the invoking thread has been released from the semaphore.

Return values

<i>RDY_OK</i>	if the binary semaphore has been successfully taken.
<i>RDY_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.

10.12.2.5 #define chBSemWaitS(*bsp*) chSemWaitS(&(*bsp*)>bs_sem)

Wait operation on the binary semaphore.

Parameters

in	<i>bsp</i>	pointer to a BinarySemaphore structure
----	------------	--

Returns

A message specifying how the invoking thread has been released from the semaphore.

Return values

<i>RDY_OK</i>	if the binary semaphore has been successfully taken.
<i>RDY_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.

10.12.2.6 #define chBSemWaitTimeout(*bsp*, *time*) chSemWaitTimeout(&(*bsp*)->bs_sem, (*time*))

Wait operation on the binary semaphore.

Parameters

in	<i>bsp</i>	pointer to a BinarySemaphore 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>RDY_OK</i>	if the binary semaphore has been successfully taken.
<i>RDY_RESET</i>	if the binary semaphore has been reset using <code>bsemReset()</code> .
<i>RDY_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.

10.12.2.7 #define chBSemWaitTimeoutS(*bsp*, *time*) chSemWaitTimeoutS(&(*bsp*)->bs_sem, (*time*))

Wait operation on the binary semaphore.

Parameters

in	<i>bsp</i>	pointer to a BinarySemaphore 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>RDY_OK</i>	if the binary semaphore has been successfully taken.
<i>RDY_RESET</i>	if the binary semaphore has been reset using <code>bsemReset()</code> .
<i>RDY_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.

10.12.2.8 #define chBSemReset(*bsp*, *taken*) chSemReset(&(*bsp*)->bs_sem, (*taken*) ? 0 : 1)

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 `bsemWait()` will return `RDY_RESET` instead of `RDY_OK`.

Parameters

in	<i>bsp</i>	pointer to a <code>BinarySemaphore</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.

```
10.12.2.9 #define chBSemResetI( bsp, taken ) chSemResetI(&(bsp)->bs_sem, (taken) ? 0 : 1)
```

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 `bsemWait()` will return `RDY_RESET` instead of `RDY_OK`.

This function does not reschedule.

Parameters

in	<i>bsp</i>	pointer to a <code>BinarySemaphore</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:

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

```
10.12.2.10 #define chBSemSignal( bsp )
```

Value:

```
{
    chSysLock();
    chBSemSignalI((bsp));
    chSchRescheduleS();
    chSysUnlock();
}
```

Performs a signal operation on a binary semaphore.

Parameters

in	<i>bsp</i>	pointer to a <code>BinarySemaphore</code> structure
----	------------	---

Function Class:

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

10.12.2.11 #define chBSemSignal(*bsp*)

Value:

```
{
    if ((bsp)->bs_sem.s_cnt < 1)
        chSemSignalI(&(bsp)->bs_sem);
}
```

Performs a signal operation on a binary semaphore.

Note

This function does not reschedule.

Parameters

in	<i>bsp</i> pointer to a BinarySemaphore 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.

10.12.2.12 #define chBSemGetState(*bsp*) ((bsp)->bs_sem.s_cnt > 0 ? FALSE : TRUE)

Returns the binary semaphore current state.

Parameters

in	<i>bsp</i> pointer to a BinarySemaphore 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.

10.13 Mutexes

10.13.1 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 are always performed in lock-reverse order. The unlock API does not even have a parameter, the mutex to unlock is selected from an internal, per-thread, stack of owned mutexes. This both improves the performance and is required for an efficient implementation of the priority inheritance mechanism.

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_USE_MUTEXES` option must be enabled in `chconf.h`.

Postcondition

Enabling mutexes requires 5-12 (depending on the architecture) extra bytes in the `Thread` structure.

Data Structures

- struct `Mutex`
Mutex structure.

Functions

- `void chMtxInit (Mutex *mp)`
Initializes a Mutex structure.
- `void chMtxLock (Mutex *mp)`
Locks the specified mutex.
- `void chMtxLockS (Mutex *mp)`
Locks the specified mutex.
- `bool_t chMtxTryLock (Mutex *mp)`
Tries to lock a mutex.
- `bool_t chMtxTryLockS (Mutex *mp)`
Tries to lock a mutex.
- `Mutex * chMtxUnlock (void)`
Unlocks the next owned mutex in reverse lock order.
- `Mutex * chMtxUnlockS (void)`
Unlocks the next owned mutex in reverse lock order.
- `void chMtxUnlockAll (void)`
Unlocks all the mutexes owned by the invoking thread.

Macro Functions

- `#define chMtxQueueNotEmptyS(mp) notempty(&(mp)->m_queue)`
Returns TRUE if the mutex queue contains at least a waiting thread.

Defines

- `#define _MUTEX_DATA(name) {_THREADSQUEUE_DATA(name.m_queue), NULL, NULL}`
Data part of a static mutex initializer.
- `#define MUTEX_DECL(name) Mutex name = _MUTEX_DATA(name)`
Static mutex initializer.

TypeDefs

- `typedef struct Mutex Mutex`
Mutex structure.

10.13.2 Function Documentation

10.13.2.1 void chMtxInit (**Mutex** * *mp*)

Initializes s **Mutex** structure.

Parameters

out *mp* pointer to a **Mutex** structure

Function Class:

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

10.13.2.2 void chMtxLock (**Mutex** * *mp*)

Locks the specified mutex.

Postcondition

The mutex is locked and inserted in the per-thread stack of owned mutexes.

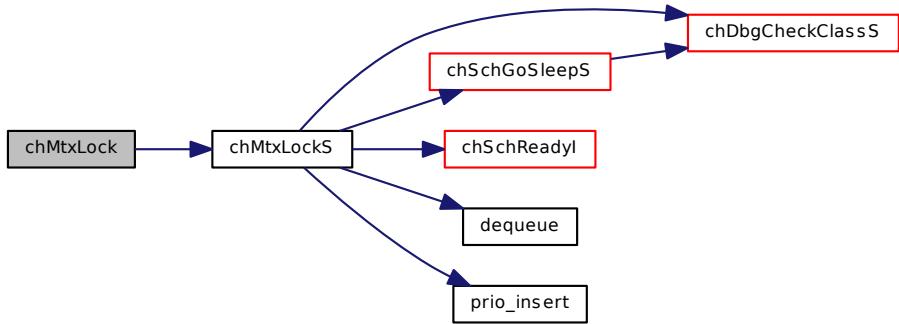
Parameters

in *mp* pointer to the **Mutex** 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:



10.13.2.3 void chMtxLockS (Mutex * mp)

Locks the specified mutex.

Postcondition

The mutex is locked and inserted in the per-thread stack of owned mutexes.

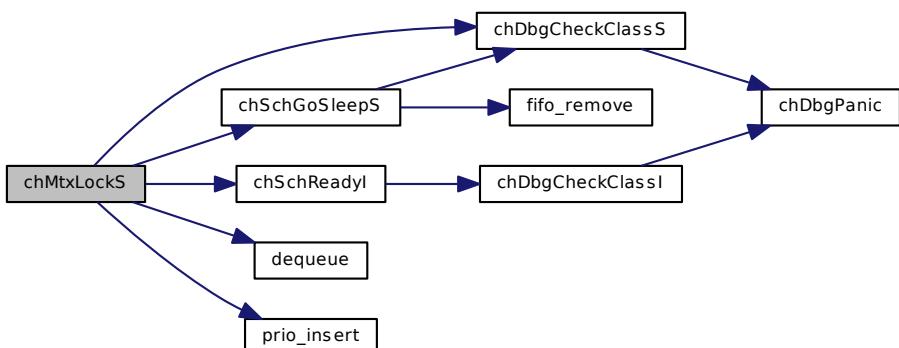
Parameters

in *mp* pointer to the [Mutex](#) 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:



10.13.2.4 `bool_t chMtxTryLock (Mutex * 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 *mp* pointer to the [Mutex](#) structure

Returns

The operation status.

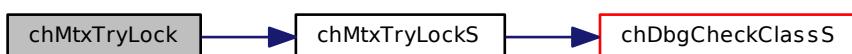
Return values

<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:



10.13.2.5 `bool_t chMtxTryLockS (Mutex * 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 *mp* pointer to the [Mutex](#) structure

Returns

The operation status.

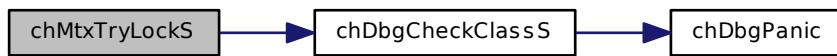
Return values

TRUE if the mutex has been successfully acquired
FALSE 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:



10.13.2.6 `Mutex *chMtxUnlock(void)`

Unlocks the next owned mutex in reverse lock order.

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.

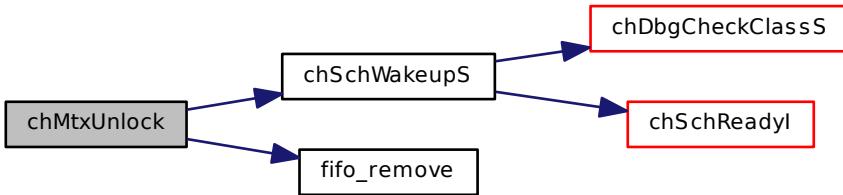
Returns

A pointer to the unlocked mutex.

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:



10.13.2.7 Mutex * chMtxUnlockS(void)

Unlocks the next owned mutex in reverse lock order.

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.

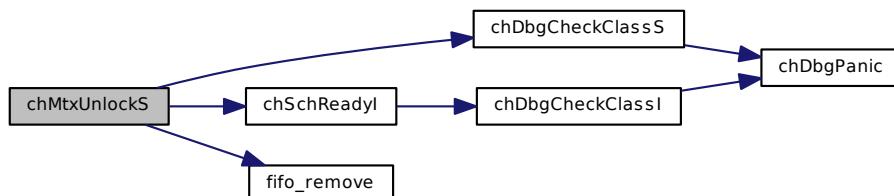
Returns

A pointer to the unlocked mutex.

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:



10.13.2.8 void chMtxUnlockAll(void)

Unlocks all the mutexes owned by the invoking thread.

Postcondition

The stack of owned mutexes is emptied and all the found mutexes are unlocked.

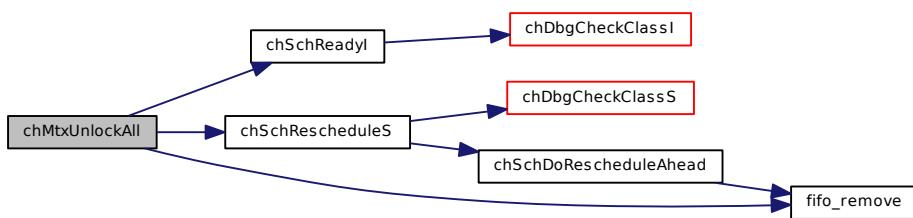
Note

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:



10.13.3 Define Documentation

10.13.3.1 `#define _MUTEX_DATA(name) { _THREADSQUEUE_DATA(name.m_queue), NULL, NULL }`

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

10.13.3.2 `#define MUTEX_DECL(name) Mutex name = _MUTEX_DATA(name)`

Static mutex initializer.

Statically initialized mutexes require no explicit initialization using `chMtxInit()`.

Parameters

in *name* the name of the mutex variable

10.13.3.3 `#define chMtxQueueNotEmptyS(mp) notempty(&(mp)->m_queue)`

Returns TRUE if the mutex queue contains at least a waiting thread.

Function Class:

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

10.13.4 Typedef Documentation

10.13.4.1 `typedef struct Mutex Mutex`

`Mutex` structure.

10.14 Condition Variables

10.14.1 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 CondVars together can implement a Monitor construct.

Precondition

In order to use the condition variable APIs the `CH_USE_CONDVAR` option must be enabled in `chconf.h`.

Data Structures

- `struct CondVar`

CondVar structure.

Functions

- `void chCondInit (CondVar *cp)`

Initializes a `CondVar` structure.

- `void chCondSignal (CondVar *cp)`

Signals one thread that is waiting on the condition variable.

- `void chCondSignall (CondVar *cp)`

Signals one thread that is waiting on the condition variable.

- `void chCondBroadcast (CondVar *cp)`

Signals all threads that are waiting on the condition variable.

- `void chCondBroadcastl (CondVar *cp)`

Signals all threads that are waiting on the condition variable.

- `msg_t chCondWait (CondVar *cp)`

Waits on the condition variable releasing the mutex lock.

- `msg_t chCondWaitS (CondVar *cp)`

Waits on the condition variable releasing the mutex lock.

- `msg_t chCondWaitTimeout (CondVar *cp, systime_t time)`

Waits on the condition variable releasing the mutex lock.

- `msg_t chCondWaitTimeoutS (CondVar *cp, systime_t time)`

Waits on the condition variable releasing the mutex lock.

Defines

- `#define _CONDVAR_DATA(name) {_THREADSQUEUE_DATA(name.c_queue)}`
Data part of a static condition variable initializer.
- `#define CONDVAR_DECL(name) CondVar name = _CONDVAR_DATA(name)`
Static condition variable initializer.

TypeDefs

- `typedef struct CondVar CondVar`
CondVar structure.

10.14.2 Function Documentation

10.14.2.1 void chCondInit (CondVar * cp)

Initializes a `CondVar` structure.

Parameters

out `cp` pointer to a `CondVar` structure

Function Class:

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

10.14.2.2 void chCondSignal (CondVar * cp)

Signals one thread that is waiting on the condition variable.

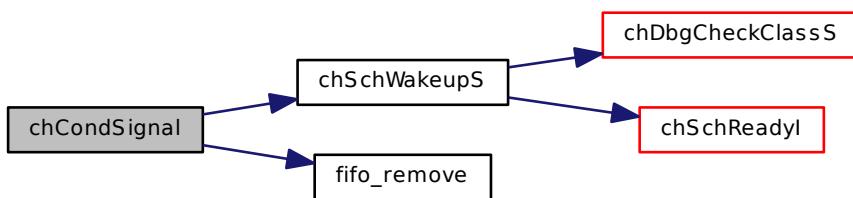
Parameters

in `cp` pointer to the `CondVar` 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:



10.14.2.3 void chCondSignal(CondVar * 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.

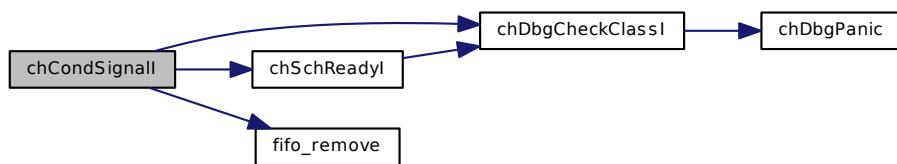
Parameters

in *cp* pointer to the [CondVar](#) 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:



10.14.2.4 void chCondBroadcast(CondVar * cp)

Signals all threads that are waiting on the condition variable.

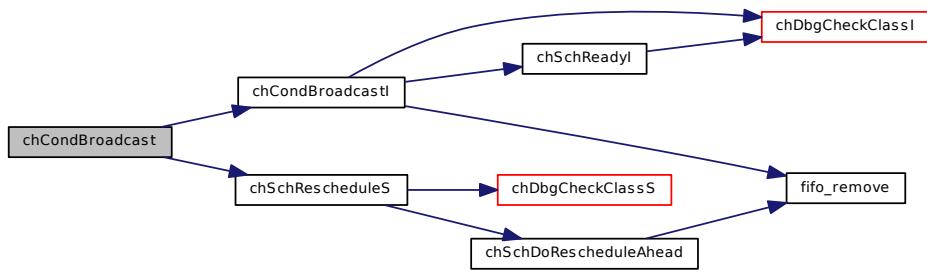
Parameters

in *cp* pointer to the [CondVar](#) 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:



10.14.2.5 void chCondBroadcastl (CondVar * 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.

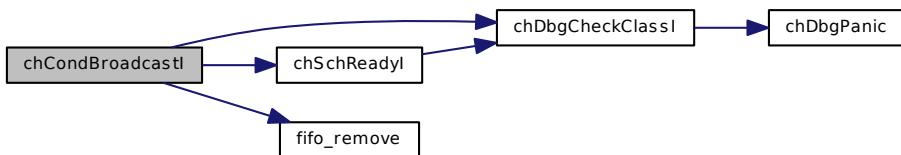
Parameters

in *cp* pointer to the [CondVar](#) 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:



10.14.2.6 msg_t chCondWait (CondVar * 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 *cp* pointer to the [CondVar](#) structure

Returns

A message specifying how the invoking thread has been released from the condition variable.

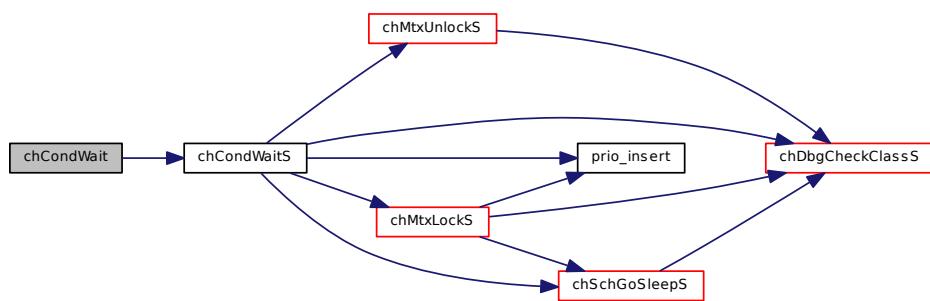
Return values

<i>RDY_OK</i>	if the condvar has been signaled using chCondSignal() .
<i>RDY_RESET</i>	if the condvar has been signaled using chCondBroadcast() .

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:

**10.14.2.7 msg_t chCondWaitS (CondVar * 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 *cp* pointer to the [CondVar](#) structure

Returns

A message specifying how the invoking thread has been released from the condition variable.

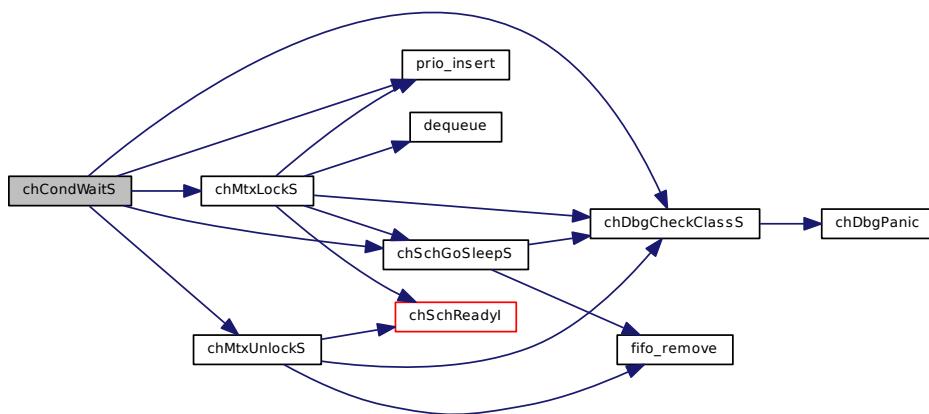
Return values

<i>RDY_OK</i>	if the condvar has been signaled using chCondSignal() .
<i>RDY_RESET</i>	if the condvar has been signaled using chCondBroadcast() .

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:



10.14.2.8 `msg_t chCondWaitTimeout (CondVar * 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_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	<code>cp</code>	pointer to the <code>CondVar</code> structure
in	<code>time</code>	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.

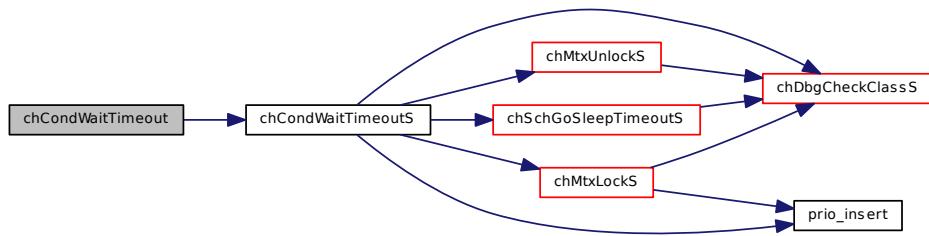
Return values

<code>RDY_OK</code>	if the condvar has been signaled using <code>chCondSignal()</code> .
<code>RDY_RESET</code>	if the condvar has been signaled using <code>chCondBroadcast()</code> .
<code>RDY_TIMEOUT</code>	if the condvar 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:

**10.14.2.9 msg_t chCondWaitTimeoutS (CondVar * 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_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	<code>cp</code>	pointer to the <code>CondVar</code> structure
in	<code>time</code>	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.

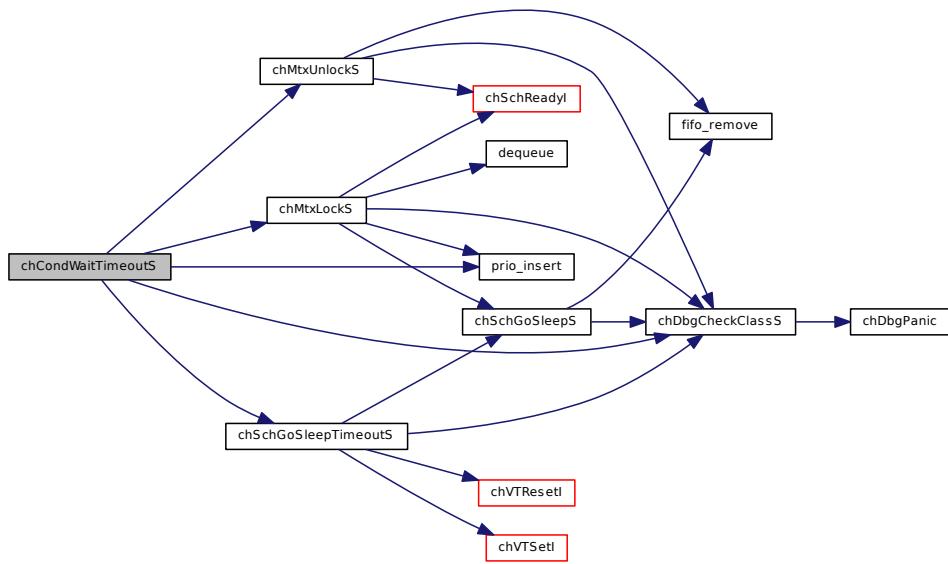
Return values

<code>RDY_OK</code>	if the condvar has been signaled using <code>chCondSignal()</code> .
<code>RDY_RESET</code>	if the condvar has been signaled using <code>chCondBroadcast()</code> .
<code>RDY_TIMEOUT</code>	if the condvar 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:



10.14.3 Define Documentation

10.14.3.1 `#define _CONDVAR_DATA(name) { _THREADSQUEUE_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

10.14.3.2 `#define CONDVAR_DECL(name) CondVar 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

10.14.4 Typedef Documentation

10.14.4.1 `typedef struct CondVar CondVar`

`CondVar` structure.

10.15 Event Flags

10.15.1 Detailed Description

Event Flags, Event Sources and Event Listeners.

Operation mode

Each thread has a mask of pending event flags inside its [Thread](#) structure. Operations defined for event flags:

- **Wait**, the invoking thread goes to sleep until a certain AND/OR combination of event flags becomes pending.
- **Clear**, a mask of event flags is cleared from the pending events mask, the cleared event flags mask is returned (only the flags that were actually pending and then cleared).
- **Signal**, an event mask is directly ORed to the mask of the signaled thread.
- **Broadcast**, each thread registered on an Event Source is signaled with the event flags 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_USE_EVENTS` option must be enabled in [chconf.h](#).

Postcondition

Enabling events requires 1-4 (depending on the architecture) extra bytes in the [Thread](#) structure.

Data Structures

- struct [EventListener](#)
Event Listener structure.
- struct [EventSource](#)
Event Source structure.

Functions

- void [chEvtRegisterMask](#) ([EventSource](#) *esp, [EventListener](#) *elp, [eventmask_t](#) mask)
Registers an Event Listener on an Event Source.
- void [chEvtUnregister](#) ([EventSource](#) *esp, [EventListener](#) *elp)
Unregisters an Event Listener from its Event Source.
- [eventmask_t](#) [chEvtGetAndClearEvents](#) ([eventmask_t](#) mask)
Clears the pending events specified in the mask.
- [eventmask_t](#) [chEvtAddEvents](#) ([eventmask_t](#) mask)
*Adds (OR) a set of event flags on the current thread, this is **much** faster than using [chEvtBroadcast\(\)](#) or [chEvtSignal\(\)](#).*
- [flagsmask_t](#) [chEvtGetAndClearFlags](#) ([EventListener](#) *elp)
Returns the flags associated to an [EventListener](#).
- [flagsmask_t](#) [chEvtGetAndClearFlagsI](#) ([EventListener](#) *elp)

- `void chEvtSignal (Thread *tp, eventmask_t mask)`

Adds a set of event flags directly to specified Thread.
- `void chEvtSignall (Thread *tp, eventmask_t mask)`

Adds a set of event flags directly to specified Thread.
- `void chEvtBroadcastFlags (EventSource *esp, flagsmask_t flags)`

Signals all the Event Listeners registered on the specified Event Source.
- `void chEvtBroadcastFlagsl (EventSource *esp, flagsmask_t flags)`

Signals all the Event Listeners registered on the specified Event Source.
- `void chEvtDispatch (const evhandler_t *handlers, eventmask_t mask)`

Invokes the event handlers associated to an event flags mask.
- `eventmask_t chEvtWaitOneTimeout (eventmask_t mask, systime_t time)`

Waits for exactly one of the specified events.
- `eventmask_t chEvtWaitAnyTimeout (eventmask_t mask, systime_t time)`

Waits for any of the specified events.
- `eventmask_t chEvtWaitAllTimeout (eventmask_t mask, systime_t time)`

Waits for all the specified events.
- `eventmask_t chEvtWaitOne (eventmask_t mask)`

Waits for exactly one of the specified events.
- `eventmask_t chEvtWaitAny (eventmask_t mask)`

Waits for any of the specified events.
- `eventmask_t chEvtWaitAll (eventmask_t mask)`

Waits for all the specified events.

Macro Functions

- `#define chEvtRegister(esp, elp, eid) chEvtRegisterMask(esp, elp, EVENT_MASK(eid))`

Registers an Event Listener on an Event Source.
- `#define chEvtInit(esp) ((esp)->es_next = (EventListener *)(void *)(esp))`

Initializes an Event Source.
- `#define chEvtIsListeningl(esp) ((void *)(esp) != (void *)(esp)->es_next)`

Verifies if there is at least one EventListener registered.
- `#define chEvtBroadcast(esp) chEvtBroadcastFlags(esp, 0)`

Signals all the Event Listeners registered on the specified Event Source.
- `#define chEvtBroadcastl(esp) chEvtBroadcastFlagsl(esp, 0)`

Signals all the Event Listeners registered on the specified Event Source.

Defines

- `#define _EVENTSOURCE_DATA(name) {(void *)&(name)}`

Data part of a static event source initializer.
- `#define EVENTSOURCE_DECL(name) EventSource name = _EVENTSOURCE_DATA(name)`

Static event source initializer.
- `#define ALL_EVENTS ((eventmask_t)-1)`

All events allowed mask.
- `#define EVENT_MASK(eid) ((eventmask_t)(1 << (eid)))`

Returns an event mask from an event identifier.

Typedefs

- **typedef struct EventSource EventSource**
Event Source structure.
- **typedef void(* evhandler_t)(eventid_t)**
Event Handler callback function.

10.15.2 Function Documentation

10.15.2.1 void chEvtRegisterMask (EventSource * esp, EventListener * elp, eventmask_t mask)

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 EventSource structure
out	<i>elp</i>	pointer to the EventListener structure
in	<i>mask</i>	the mask of event flags 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.

10.15.2.2 void chEvtUnregister (EventSource * esp, EventListener * 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 EventSource structure
in	<i>elp</i>	pointer to the EventListener structure

Function Class:

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

10.15.2.3 eventmask_t chEvtGetAndClearEvents (eventmask_t mask)

Clears the pending events specified in the mask.

Parameters

in	<i>mask</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.

10.15.2.4 eventmask_t chEvtAddEvents (eventmask_t mask)

Adds (OR) a set of event flags on the current thread, this is **much** faster than using [chEvtBroadcast\(\)](#) or [chEvtSignal\(\)](#).

Parameters

in *mask* the event flags to be added

Returns

The current pending events mask.

Function Class:

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

10.15.2.5 flagsmask_t chEvtGetAndClearFlags (EventListener * elp)

Returns the flags associated to an [EventListener](#).

The flags are returned and the [EventListener](#) flags mask is cleared.

Parameters

in *elp* pointer to the [EventListener](#) 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.

10.15.2.6 flagsmask_t chEvtGetAndClearFlagsI (EventListener * elp)

Returns the flags associated to an [EventListener](#).

The flags are returned and the [EventListener](#) flags mask is cleared.

Parameters

in *elp* pointer to the [EventListener](#) 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.

10.15.2.7 void chEvtSignal (Thread * *tp*, eventmask_t *mask*)

Adds a set of event flags directly to specified [Thread](#).

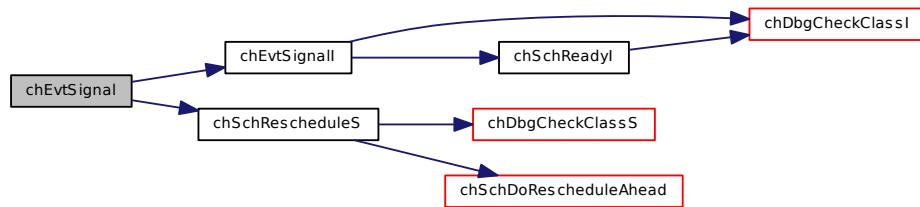
Parameters

in	<i>tp</i>	the thread to be signaled
in	<i>mask</i>	the event flags 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:



10.15.2.8 void chEvtSignalI (Thread * *tp*, eventmask_t *mask*)

Adds a set of event flags directly to specified [Thread](#).

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.

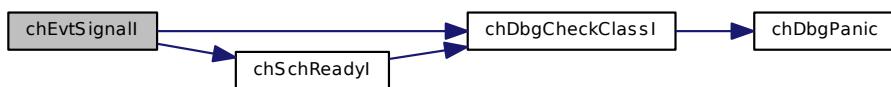
Parameters

in	<i>tp</i>	the thread to be signaled
in	<i>mask</i>	the event flags 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:



10.15.2.9 void chEvtBroadcastFlags (EventSource * esp, flagsmask_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 `EventSource` in addition to the event flags specified by the threads themselves in the `EventListener` objects.

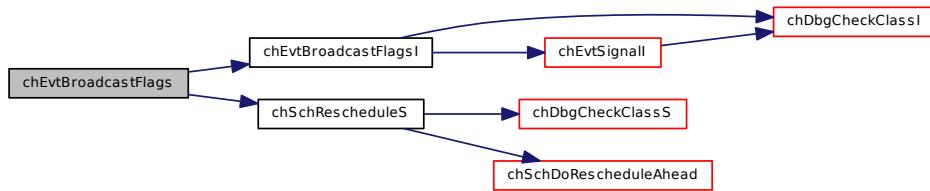
Parameters

in	<code>esp</code>	pointer to the <code>EventSource</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:



10.15.2.10 void chEvtBroadcastFlagsI (EventSource * esp, flagsmask_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 `EventSource` in addition to the event flags specified by the threads themselves in the `EventListener` 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.

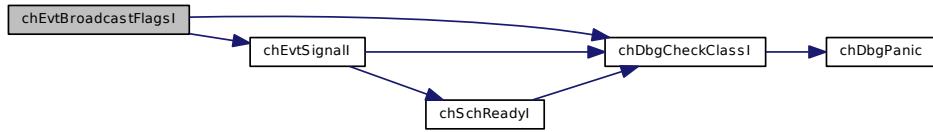
Parameters

in	<code>esp</code>	pointer to the <code>EventSource</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:



10.15.2.11 void chEvtDispatch (const evhandler_t * handlers, eventmask_t mask)

Invokes the event handlers associated to an event flags mask.

Parameters

in	<i>mask</i>	mask of the event flags to be dispatched
in	<i>handlers</i>	an array of evhandler_t. The array must have size equal to the number of bits in eventmask_t.

Function Class:

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

10.15.2.12 eventmask_t chEvtWaitOneTimeout (eventmask_t mask, systime_t time)

Waits for exactly one of the specified events.

The function waits for one event among those specified in *mask* 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	<i>mask</i>	mask of the event flags that the function should wait for, ALL_EVENTS enables all the events
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

The mask of the lowest id served and cleared event.

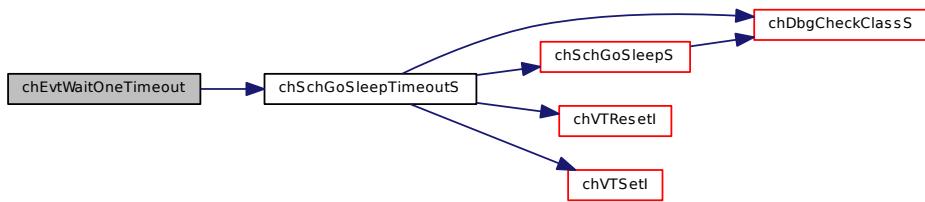
Return values

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:

**10.15.2.13 eventmask_t chEvtWaitAnyTimeout (eventmask_t mask, systime_t time)**

Waits for any of the specified events.

The function waits for any event among those specified in `mask` to become pending then the events are cleared and returned.

Parameters

in	<code>mask</code>	mask of the event flags that the function should wait for, <code>ALL_EVENTS</code> enables all the events
in	<code>time</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 mask of the served and cleared events.

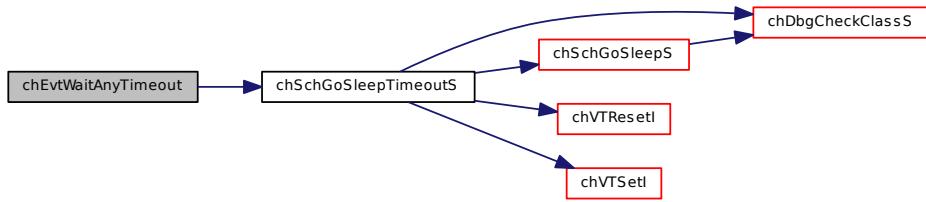
Return values

`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:



10.15.2.14 eventmask_t chEvtWaitAllTimeout (eventmask_t *mask*, systime_t *time*)

Waits for all the specified events.

The function waits for all the events specified in *mask* to become pending then the events are cleared and returned.

Parameters

in	<i>mask</i>	mask of the event flags that the function should wait for, ALL_EVENTS requires all the events
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

The mask of the served and cleared events.

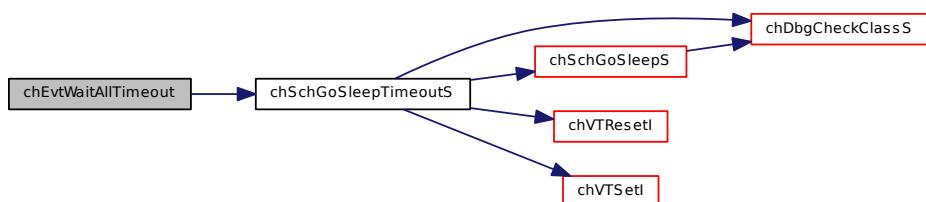
Return values

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:



10.15.2.15 eventmask_t chEvtWaitOne (eventmask_t mask)

Waits for exactly one of the specified events.

The function waits for one event among those specified in `mask` 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 `mask` mask of the event flags that the function should wait for, `ALL_EVENTS` enables all the events

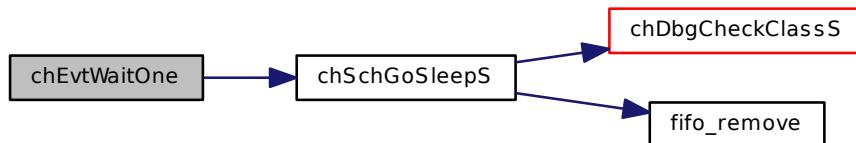
Returns

The mask of the lowest id served and cleared event.

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:



10.15.2.16 eventmask_t chEvtWaitAny (eventmask_t mask)

Waits for any of the specified events.

The function waits for any event among those specified in `mask` to become pending then the events are cleared and returned.

Parameters

in `mask` mask of the event flags that the function should wait for, `ALL_EVENTS` 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:



10.15.2.17 eventmask_t chEvtWaitAll(eventmask_t mask)

Waits for all the specified events.

The function waits for all the events specified in `mask` to become pending then the events are cleared and returned.

Parameters

in `mask` mask of the event flags that the function should wait for, `ALL_EVENTS` requires all the events

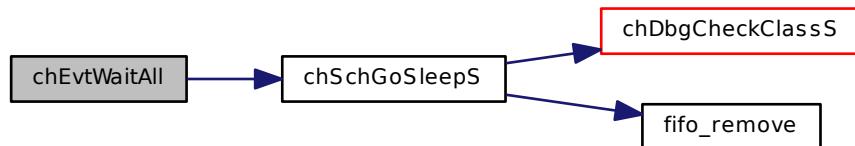
Returns

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:



10.15.3 Define Documentation

10.15.3.1 #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

name the name of the event source variable

10.15.3.2 #define EVENTSOURCE_DECL(*name*) EventSource name = _EVENTSOURCE_DATA(*name*)

Static event source initializer.

Statically initialized event sources require no explicit initialization using `chEvtInit()`.

Parameters

name the name of the event source variable

10.15.3.3 #define ALL_EVENTS ((eventmask_t)-1)

All events allowed mask.

10.15.3.4 #define EVENT_MASK(*eid*) ((eventmask_t)(1 << (*eid*)))

Returns an event mask from an event identifier.

10.15.3.5 #define chEvtRegister(*esp*, *elp*, *eid*) chEvtRegisterMask(*esp*, *elp*, EVENT_MASK(*eid*))

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>EventSource</code> structure
out	<i>elp</i>	pointer to the <code>EventListener</code> structure
in	<i>eid</i>	numeric identifier assigned to the Event Listener. The identifier is used as index for the event callback function. 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.

10.15.3.6 #define chEvtInit(*esp*) ((*esp*)->es_next = (`EventListener` *)(void *)(*esp*))

Initializes an Event Source.

Note

This function can be invoked before the kernel is initialized because it just prepares a `EventSource` structure.

Parameters

out	<i>esp</i>	pointer to the <code>EventSource</code> structure
-----	------------	---

Function Class:

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

```
10.15.3.7 #define chEvtIsListening( esp ) ((void *)esp) != (void *)(esp)->es_next
```

Verifies if there is at least one [Event Listener](#) registered.

Parameters

in	<code>esp</code> pointer to the EventSource 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.

```
10.15.3.8 #define chEvtBroadcast( esp ) chEvtBroadcastFlags(esp, 0)
```

Signals all the Event Listeners registered on the specified Event Source.

Parameters

in	<code>esp</code> pointer to the EventSource structure
----	---

Function Class:

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

```
10.15.3.9 #define chEvtBroadcastI( esp ) chEvtBroadcastFlagsI(esp, 0)
```

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	<code>esp</code> pointer to the EventSource 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.

10.15.4 Typedef Documentation

10.15.4.1 `typedef struct EventSource EventSource`

Event Source structure.

10.15.4.2 `typedef void(* evhandler_t)(eventid_t)`

Event Handler callback function.

10.16 Synchronous Messages

10.16.1 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_USE_MESSAGES_PRIORITY` option in `chconf.h`.

Precondition

In order to use the message APIs the `CH_USE_MESSAGES` option must be enabled in `chconf.h`.

Postcondition

Enabling messages requires 6-12 (depending on the architecture) extra bytes in the `Thread` structure.

Functions

- `msg_t chMsgSend (Thread *tp, msg_t msg)`
Sends a message to the specified thread.
- `Thread * chMsgWait (void)`
Suspends the thread and waits for an incoming message.
- `void chMsgRelease (Thread *tp, msg_t msg)`
Releases a sender thread specifying a response message.

Macro Functions

- `#define chMsgIsPending(tp) ((tp)->p_msgqueue.p_next != (Thread *)(&(tp))->p_msgqueue)`
Evaluates to TRUE if the thread has pending messages.
- `#define chMsgGet(tp) ((tp)->p_msg)`
Returns the message carried by the specified thread.
- `#define chMsgReleaseS(tp, msg) chSchWakeupS(tp, msg)`
Releases the thread waiting on top of the messages queue.

10.16.2 Function Documentation

10.16.2.1 `msg_t chMsgSend (Thread * 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	<code>msg</code>	the message

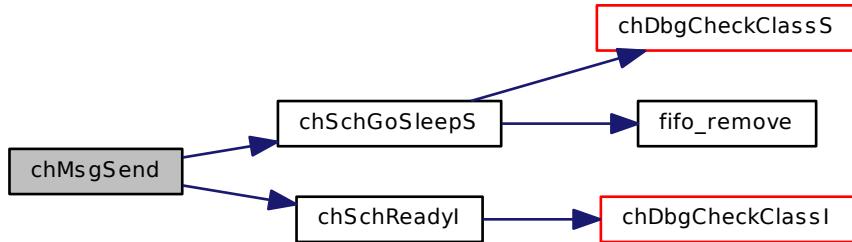
Returns

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:

**10.16.2.2 Thread * 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.

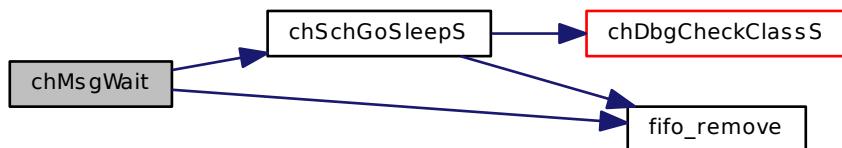
Returns

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:



10.16.2.3 void chMsgRelease (Thread * *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.

10.16.3 Define Documentation**10.16.3.1 #define chMsgIsPending(*tp*) ((tp)->p_msgqueue.p_next != (Thread *)&(tp)->p_msgqueue)**

Evaluates to TRUE if the thread has pending 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.

10.16.3.2 #define chMsgGet(*tp*) ((tp)->p_msg)

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.

10.16.3.3 #define chMsgReleaseS(*tp*, *msg*) chSchWakeupS(*tp*, *msg*)

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.

10.17 Mailboxes

10.17.1 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_USE_MAILBOXES` option must be enabled in `chconf.h`.

Data Structures

- struct `Mailbox`

Structure representing a mailbox object.

Functions

- void `chMBInit (Mailbox *mbp, msg_t *buf, cnt_t n)`
Initializes a `Mailbox` object.
- void `chMBReset (Mailbox *mbp)`
Resets a `Mailbox` object.
- `msg_t chMBPost (Mailbox *mbp, msg_t msg, systime_t time)`
Posts a message into a mailbox.
- `msg_t chMBPostS (Mailbox *mbp, msg_t msg, systime_t time)`
Posts a message into a mailbox.
- `msg_t chMBPostI (Mailbox *mbp, msg_t msg)`
Posts a message into a mailbox.
- `msg_t chMBPostAhead (Mailbox *mbp, msg_t msg, systime_t time)`
Posts an high priority message into a mailbox.
- `msg_t chMBPostAheadS (Mailbox *mbp, msg_t msg, systime_t time)`
Posts an high priority message into a mailbox.

- `msg_t chMBPostAheadI (Mailbox *mbp, msg_t msg)`
Posts an high priority message into a mailbox.
- `msg_t chMBFetch (Mailbox *mbp, msg_t *msgp, systime_t time)`
Retrieves a message from a mailbox.
- `msg_t chMBFetchS (Mailbox *mbp, msg_t *msgp, systime_t time)`
Retrieves a message from a mailbox.
- `msg_t chMBFetchI (Mailbox *mbp, msg_t *msgp)`
Retrieves a message from a mailbox.

Macro Functions

- `#define chMBSizel(mbp) ((mbp)->mb_top - (mbp)->mb_buffer)`
Returns the mailbox buffer size.
- `#define chMBGetFreeCountl(mbp) chSemGetCounterl(&(mbp)->mb_emptysem)`
Returns the number of free message slots into a mailbox.
- `#define chMBGetUsedCountl(mbp) chSemGetCounterl(&(mbp)->mb_fullsem)`
Returns the number of used message slots into a mailbox.
- `#define chMBPeekl(mbp) (*(mbp)->mb_rdptr)`
Returns the next message in the queue without removing it.

Defines

- `#define _MAILBOX_DATA(name, buffer, size)`
Data part of a static mailbox initializer.
- `#define MAILBOX_DECL(name, buffer, size) Mailbox name = _MAILBOX_DATA(name, buffer, size)`
Static mailbox initializer.

10.17.2 Function Documentation

10.17.2.1 void chMBInit (Mailbox * mbp, msg_t * buf, cnt_t n)

Initializes a `Mailbox` object.

Parameters

out	<code>mbp</code>	the pointer to the <code>Mailbox</code> structure to be initialized
in	<code>buf</code>	pointer to the messages buffer as an array of <code>msg_t</code>
in	<code>n</code>	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:



10.17.2.2 void chMBReset (Mailbox * mbp)

Resets a [Mailbox](#) object.

All the waiting threads are resumed with status `RDY_RESET` and the queued messages are lost.

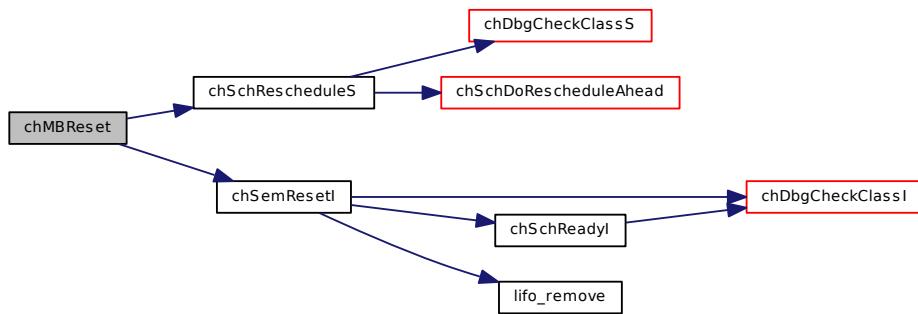
Parameters

in	<i>mbp</i> the pointer to an initialized Mailbox 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:



10.17.2.3 msg_t chMBPost (Mailbox * mbp, msg_t msg, systime_t time)

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 object
in	<i>msg</i> the message to be posted on the mailbox
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

The operation status.

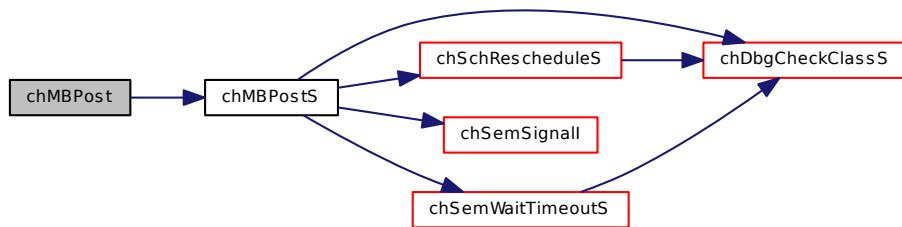
Return values

<code>RDY_OK</code>	if a message has been correctly posted.
<code>RDY_RESET</code>	if the mailbox has been reset while waiting.
<code>RDY_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:



10.17.2.4 `msg_t chMBPostS(Mailbox *mbp, msg_t msg, systime_t time)`

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 Mailbox object
in	<code>msg</code>	the message to be posted on the mailbox
in	<code>time</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.

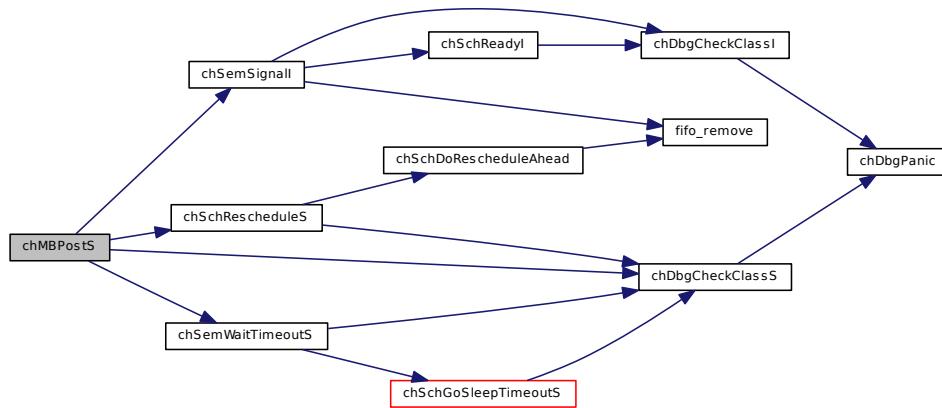
Return values

<code>RDY_OK</code>	if a message has been correctly posted.
<code>RDY_RESET</code>	if the mailbox has been reset while waiting.
<code>RDY_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:



10.17.2.5 msg_t chMBPostI (Mailbox * 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 Mailbox object
in	<code>msg</code>	the message to be posted on the mailbox

Returns

The operation status.

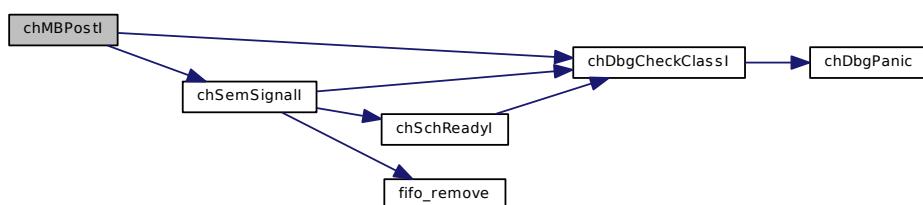
Return values

<code>RDY_OK</code>	if a message has been correctly posted.
<code>RDY_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:



10.17.2.6 `msg_t chMBPostAhead (Mailbox * mbp, msg_t msg, systime_t time)`

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 Mailbox object
in	<i>msg</i>	the message to be posted on the mailbox
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

The operation status.

Return values

RDY_OK if a message has been correctly posted.

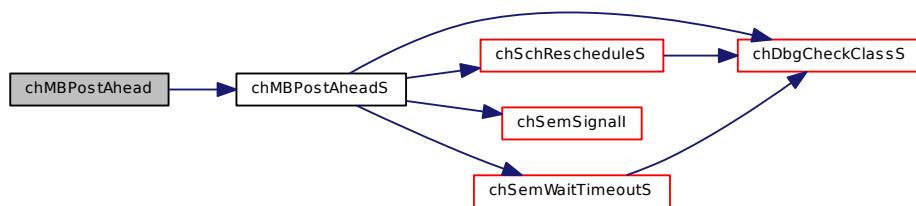
RDY_RESET if the mailbox has been reset while waiting.

RDY_TIMEOUT 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:



10.17.2.7 `msg_t chMBPostAheadS (Mailbox * mbp, msg_t msg, systime_t time)`

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 Mailbox object
in	<i>msg</i>	the message to be posted on the mailbox
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

The operation status.

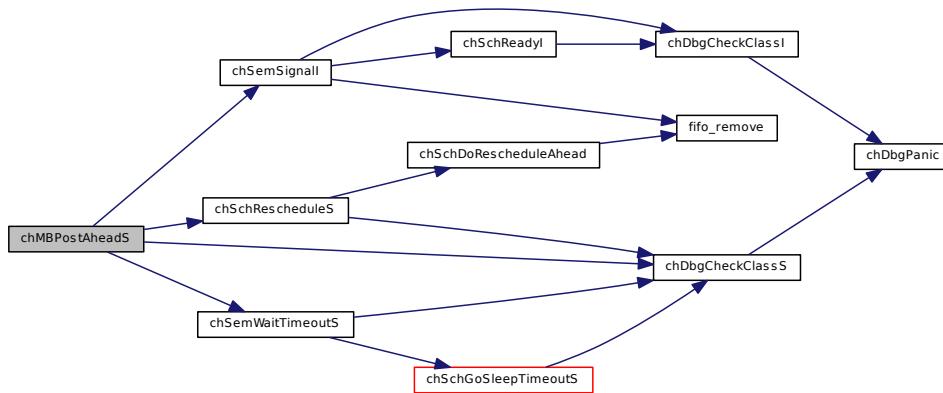
Return values

- RDY_OK* if a message has been correctly posted.
- RDY_RESET* if the mailbox has been reset while waiting.
- RDY_TIMEOUT* 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:

**10.17.2.8 msg_t chMBPostAheadI (Mailbox * 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 object |
| in | <i>msg</i> | the message to be posted on the mailbox |

Returns

The operation status.

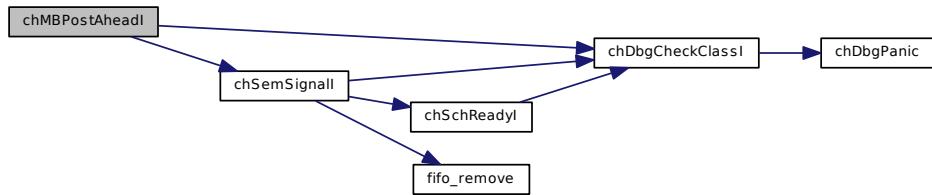
Return values

- RDY_OK* if a message has been correctly posted.
- RDY_TIMEOUT* 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:



10.17.2.9 `msg_t chMBFetch (Mailbox *mbp, msg_t *msgp, systime_t time)`

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 Mailbox object
out	<code>msgp</code>	pointer to a message variable for the received message
in	<code>time</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.

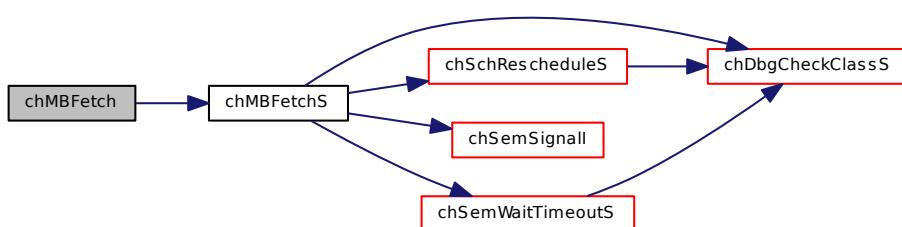
Return values

<code>RDY_OK</code>	if a message has been correctly fetched.
<code>RDY_RESET</code>	if the mailbox has been reset while waiting.
<code>RDY_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:



10.17.2.10 `msg_t chMBFetchS (Mailbox * mbp, msg_t * msgp, systime_t time)`

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	<i>mbp</i>	the pointer to an initialized Mailbox object
out	<i>msgp</i>	pointer to a message variable for the received message
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

The operation status.

Return values

RDY_OK if a message has been correctly fetched.

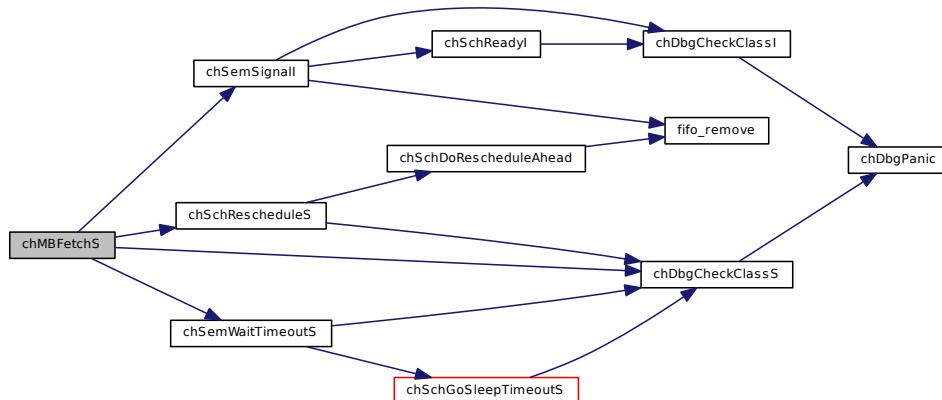
RDY_RESET if the mailbox has been reset while waiting.

RDY_TIMEOUT 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:



10.17.2.11 `msg_t chMBFetchI (Mailbox * 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

in	<i>mbp</i>	the pointer to an initialized Mailbox object
out	<i>msgp</i>	pointer to a message variable for the received message

Returns

The operation status.

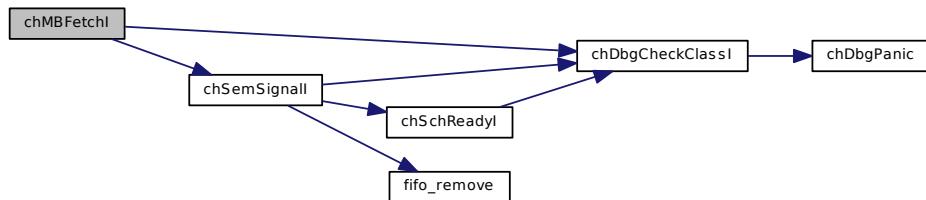
Return values

<i>RDY_OK</i>	if a message has been correctly fetched.
<i>RDY_TIMEOUT</i>	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:

**10.17.3 Define Documentation**

10.17.3.1 `#define chMBSizel(mbp) ((mbp)->mb_top - (mbp)->mb_buffer)`

Returns the mailbox buffer size.

Parameters

in *mbp* the pointer to an initialized [Mailbox](#) object

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.

10.17.3.2 `#define chMBGetFreeCountl(mbp) chSemGetCounterl(&(mbp)->mb_emptysem)`

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 *mbp* the pointer to an initialized [Mailbox](#) 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.

```
10.17.3.3 #define chMBGetUsedCountI( mbp ) chSemGetCounterI(&(mbp)->mb_fullsem)
```

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 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.

```
10.17.3.4 #define chMBPeekI( mbp ) (*(mbp)->mb_rdptr)
```

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.

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.

```
10.17.3.5 #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	<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

```
10.17.3.6 #define MAILBOX_DECL( name, buffer, size ) Mailbox 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

10.18 I/O Queues

10.18.1 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_USE_QUEUES` option must be enabled in `chconf.h`.

Data Structures

- struct `GenericQueue`
Generic I/O queue structure.

Functions

- void `chIQInit (InputQueue *iqp, uint8_t *bp, size_t size, qnotify_t infy, void *link)`
Initializes an input queue.
- void `chIQResetI (InputQueue *iqp)`
Resets an input queue.
- msg_t `chIQPutI (InputQueue *iqp, uint8_t b)`
Input queue write.
- msg_t `chIQGetTimeout (InputQueue *iqp, systime_t time)`
Input queue read with timeout.
- size_t `chIQReadTimeout (InputQueue *iqp, uint8_t *bp, size_t n, systime_t time)`

- **Input queue read with timeout.**
- **void chOQInit (OutputQueue *oqp, uint8_t *bp, size_t size, qnotify_t onfy, void *link)**
Initializes an output queue.
- **void chOQResetI (OutputQueue *oqp)**
Resets an output queue.
- **msg_t chOQPutTimeout (OutputQueue *oqp, uint8_t b, systime_t time)**
Output queue write with timeout.
- **msg_t chOQGetI (OutputQueue *oqp)**
Output queue read.
- **size_t chOQWriteTimeout (OutputQueue *oqp, const uint8_t *bp, size_t n, systime_t time)**
Output queue write with timeout.

Queue functions returned status value

- **#define Q_OK RDY_OK**
Operation successful.
- **#define Q_TIMEOUT RDY_TIMEOUT**
Timeout condition.
- **#define Q_RESET RDY_RESET**
Queue has been reset.
- **#define Q_EMPTY -3**
Queue empty.
- **#define Q_FULL -4**
Queue full.

Macro Functions

- **#define chQSizeI(qp) ((size_t)((qp)->q_top - (qp)->q_buffer))**
Returns the queue's buffer size.
- **#define chQSpaceI(qp) ((qp)->q_counter)**
Queue space.
- **#define chQGetLink(qp) ((qp)->q_link)**
Returns the queue application-defined link.
- **#define chIQGetFullI(iqp) chQSpaceI(iqp)**
Returns the filled space into an input queue.
- **#define chIQGetEmptyI(iqp) (chQSizeI(iqp) - chQSpaceI(iqp))**
Returns the empty space into an input queue.
- **#define chIQIsEmptyI(iqp) ((bool_t)(chQSpaceI(iqp) <= 0))**
Evaluates to TRUE if the specified input queue is empty.
- **#define chIQIsFullI(iqp)**
Evaluates to TRUE if the specified input queue is full.
- **#define chIQGet(iqp) chIQGetTimeout(iqp, TIME_INFINITE)**
Input queue read.
- **#define chOQGetFullI(oqp) (chQSizeI(oqp) - chQSpaceI(oqp))**
Returns the filled space into an output queue.
- **#define chOQGetEmptyI(oqp) chQSpaceI(oqp)**
Returns the empty space into an output queue.
- **#define chOQIsEmptyI(oqp)**
Evaluates to TRUE if the specified output queue is empty.
- **#define chOQIsFullI(oqp) ((bool_t)(chQSpaceI(oqp) <= 0))**
Evaluates to TRUE if the specified output queue is full.
- **#define chOQPut(oqp, b) chOQPutTimeout(oqp, b, TIME_INFINITE)**
Output queue write.

Defines

- `#define _INPUTQUEUE_DATA(name, buffer, size, inotify, link)`
Data part of a static input queue initializer.
- `#define INPUTQUEUE_DECL(name, buffer, size, inotify, link) InputQueue 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) OutputQueue name = _OUTPUTQUEUE_DATA(name, buffer, size, onotify, link)`
Static output queue initializer.

TypeDefs

- `typedef struct GenericQueue GenericQueue`
Type of a generic I/O queue structure.
- `typedef void(* qnotify_t)(GenericQueue *qp)`
Queue notification callback type.
- `typedef GenericQueue InputQueue`
Type of an input queue structure.
- `typedef GenericQueue OutputQueue`
Type of an output queue structure.

10.18.2 Function Documentation

10.18.2.1 void chIQInit (InputQueue * *iqp*, uint8_t * *bp*, size_t *size*, qnotify_t *infty*, 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

out	<i>iqp</i>	pointer to an <code>InputQueue</code> structure
in	<i>bp</i>	pointer to a memory area allocated as queue buffer
in	<i>size</i>	size of the queue buffer
in	<i>infty</i>	pointer to a callback function that is invoked when data is read from the queue. The value can be <code>NULL</code> .
in	<i>link</i>	application defined pointer

Function Class:

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

10.18.2.2 void chIQResetl (InputQueue * *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.

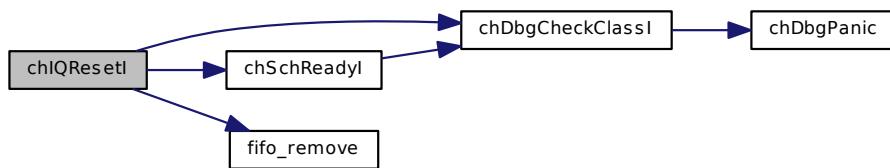
Parameters

in	<i>iqp</i> pointer to an <code>InputQueue</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:

**10.18.2.3 msg_t chIQPutI (InputQueue * *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>InputQueue</code> structure
in	<i>b</i> the byte value to be written in the queue

Returns

The operation status.

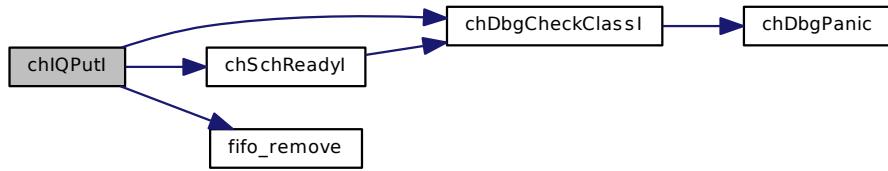
Return values

<code>Q_OK</code>	if the operation has been completed with success.
<code>Q_FULL</code>	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:



10.18.2.4 msg_t chIQGetTimeout (InputQueue * *iqp*, systime_t *time*)

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 THD_STATE_WTQUEUE.

Parameters

in	<i>iqp</i>	pointer to an <code>InputQueue</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 byte value from the queue.

Return values

<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.

10.18.2.5 size_t chIQReadTimeout (InputQueue * *iqp*, uint8_t * *bp*, size_t *n*, systime_t *time*)

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 THD_STATE_WTQUEUE.

Parameters

in	<i>iqp</i>	pointer to an InputQueue 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>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

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.

10.18.2.6 void chOQInit (OutputQueue * *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

out	<i>oqp</i>	pointer to an OutputQueue structure
in	<i>bp</i>	pointer to a memory area allocated as queue buffer
in	<i>size</i>	size of the queue buffer
in	<i>onfy</i>	pointer to a callback function that is invoked when data is written to the queue. The value can be NULL.
in	<i>link</i>	application defined pointer

Function Class:

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

10.18.2.7 void chOQReset (OutputQueue * *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.

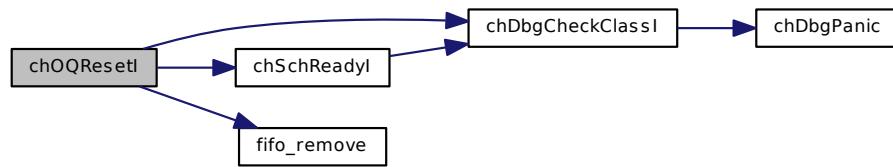
Parameters

in	<i>oqp</i>	pointer to an OutputQueue 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:



10.18.2.8 `msg_t chOQPutTimeout (OutputQueue * oqp, uint8_t b, systime_t time)`

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

<code>in</code>	<code>oqp</code>	pointer to an <code>OutputQueue</code> structure
<code>in</code>	<code>b</code>	the byte value to be written in the queue
<code>in</code>	<code>time</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>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.

10.18.2.9 `msg_t chOQGetI (OutputQueue * oqp)`

Output queue read.

A byte value is read from the low end of an output queue.

Parameters

in *oqp* pointer to an `OutputQueue` structure

Returns

The byte value from the queue.

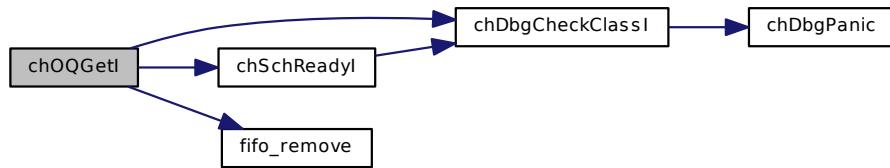
Return values

Q_EMPTY 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:

**10.18.2.10 size_t chOQWriteTimeout (OutputQueue * *oqp*, const uint8_t * *bp*, size_t *n*, systime_t *time*)**

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 *oqp* pointer to an `OutputQueue` structure

in *bp* pointer to the data buffer

in *n* the maximum amount of data to be transferred, the value 0 is reserved

in *time* the number of ticks before the operation timeouts, the following special values are allowed:

- *TIME_IMMEDIATE* immediate timeout.
- *TIME_INFINITE* 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.

10.18.3 Define Documentation

10.18.3.1 #define Q_OK RDY_OK

Operation successful.

10.18.3.2 #define Q_TIMEOUT RDY_TIMEOUT

Timeout condition.

10.18.3.3 #define Q_RESET RDY_RESET

Queue has been reset.

10.18.3.4 #define Q_EMPTY -3

Queue empty.

10.18.3.5 #define Q_FULL -4

Queue full.,

10.18.3.6 #define chQSize(*qp*) ((size_t)((*qp*)>q_top - (*qp*)>q_buffer))

Returns the queue's buffer size.

Parameters

in *qp* pointer to a [GenericQueue](#) structure.

Returns

The buffer size.

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.

10.18.3.7 #define chQSpace(*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 *qp* pointer to a [GenericQueue](#) 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.

10.18.3.8 #define chQGetLink(*qp*) ((*qp*)->q_link)

Returns the queue application-defined link.

Note

This function can be called in any context.

Parameters

in *qp* pointer to a [GenericQueue](#) structure.

Returns

The application-defined link.

Function Class:

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

10.18.3.9 #define chIQGetFulll(*iqp*) chQSpaceI(*iqp*)

Returns the filled space into an input queue.

Parameters

in *iqp* pointer to an [InputQueue](#) structure

Returns

The number of full bytes in the queue.

Return values

0 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.

10.18.3.10 #define chIQGetEmptyl(*iqp*) (chQSizeI(*iqp*) - chQSpaceI(*iqp*))

Returns the empty space into an input queue.

Parameters

in *iqp* pointer to an [InputQueue](#) 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.

10.18.3.11 #define chIQIsEmpty(*iqp*) ((bool_t)(chQSpace(iqp) <= 0))

Evaluates to TRUE if the specified input queue is empty.

Parameters

in *iqp* pointer to an `InputQueue` structure.

Returns

The queue status.

Return values

FALSE if the queue is not empty.

TRUE 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.

10.18.3.12 #define chIQIsFull(*iqp*)

Value:

```
((bool_t) (((iqp)->q_wrptr == (iqp)->q_rdptr) && \
           ((iqp)->q_counter != 0)))
```

Evaluates to TRUE if the specified input queue is full.

Parameters

in *iqp* pointer to an `InputQueue` structure.

Returns

The queue status.

Return values

FALSE if the queue is not full.

TRUE 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.

10.18.3.13 #define chIQGet(*iqp*) chIQGetTimeout(iqp, TIME_INFINITE)

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 *iqp* pointer to an `InputQueue` structure

Returns

A byte value from the queue.

Return values

Q_RESET 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.

```
10.18.3.14 #define _INPUTQUEUE_DATA( name, buffer, size, inotify, link )
```

Value:

```
{
    \
    _THREADSQUEUE_DATA(name),
    \
    0,
    \
    (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

```
10.18.3.15 #define INPUTQUEUE_DECL( name, buffer, size, inotify, link ) InputQueue 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

```
10.18.3.16 #define chOQGetFull( oqp ) (chQSize(oqp) - chQSpace(oqp))
```

Returns the filled space into an output queue.

Parameters

in *oqp* pointer to an `OutputQueue` structure

Returns

The number of full bytes in the queue.

Return values

O 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.

10.18.3.17 #define chOQGetEmpty(*oqp*) chQSpace(*oqp*)

Returns the empty space into an output queue.

Parameters

in *oqp* pointer to an `OutputQueue` structure

Returns

The number of empty bytes in the queue.

Return values

O 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.

10.18.3.18 #define chOQIsEmpty(*oqp*)

Value:

```
((bool_t)((oqp)->q_wrptr == (oqp)->q_rdptr) && \
((oqp)->q_counter != 0)))
```

Evaluates to TRUE if the specified output queue is empty.

Parameters

in *oqp* pointer to an `OutputQueue` structure.

Returns

The queue status.

Return values

FALSE if the queue is not empty.

TRUE 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.

10.18.3.19 #define chOQIsFull(*oqp*) ((bool_t)(chQSpace(*oqp*) <= 0))

Evaluates to TRUE if the specified output queue is full.

Parameters

in	<i>oqp</i> pointer to an OutputQueue 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.

10.18.3.20 #define chOQPut(*oqp*, *b*) chOQPutTimeout(*oqp*, *b*, TIME_INFINITE)

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 OutputQueue 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 succeeded.
<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.

10.18.3.21 #define _OUTPUTQUEUE_DATA(*name*, *buffer*, *size*, *onotify*, *link*)

Value:

```
{
    \
    _THREADSQUEUE_DATA(name),
    \
    (size_t),
    \
    (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

```
10.18.3.22 #define OUTPUTQUEUE_DECL( name, buffer, size, onotify, link ) OutputQueue 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

10.18.4 Typedef Documentation

10.18.4.1 **typedef struct GenericQueue GenericQueue**

Type of a generic I/O queue structure.

10.18.4.2 **typedef void(* qnotify_t)(GenericQueue *qp)**

Queue notification callback type.

10.18.4.3 **typedef GenericQueue InputQueue**

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.

10.18.4.4 **typedef GenericQueue OutputQueue**

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.

10.19 Memory Management

10.19.1 Detailed Description

Memory Management services.

Modules

- [Core Memory Manager](#)
- [Heaps](#)
- [Memory Pools](#)
- [Dynamic Threads](#)

10.20 Core Memory Manager

10.20.1 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_USE_MEMCORE` option must be enabled in [`chconf.h`](#).

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 chCoreStatus (void)`
Core memory status.

Alignment support macros

- `#define MEM_ALIGN_SIZE sizeof(stkalign_t)`
Alignment size constant.
- `#define MEM_ALIGN_MASK (MEM_ALIGN_SIZE - 1)`
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) == 0)`
Returns whatever a pointer or memory size is aligned to the type align_t.

Typedefs

- `typedef void *(*memgetfunc_t)(size_t size)`
Memory get function.

10.20.2 Function Documentation

10.20.2.1 void _core_init(void)

Low level memory manager initialization.

Function Class:

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

10.20.2.2 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

`NULL` 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:



10.20.2.3 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.

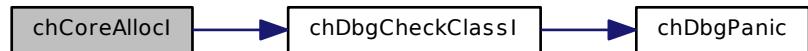
Return values

`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:



10.20.2.4 size_t chCoreStatus(void)

Core memory status.

Returns

The size, in bytes, of the free core memory.

Function Class:

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

10.20.3 Define Documentation

10.20.3.1 `#define MEM_ALIGN_SIZE sizeof(stkalign_t)`

Alignment size constant.

10.20.3.2 `#define MEM_ALIGN_MASK (MEM_ALIGN_SIZE - 1)`

Alignment mask constant.

10.20.3.3 `#define MEM_ALIGN_PREV(p) ((size_t)(p) & ~MEM_ALIGN_MASK)`

Alignment helper macro.

10.20.3.4 `#define MEM_ALIGN_NEXT(p) MEM_ALIGN_PREV((size_t)(p) + MEM_ALIGN_MASK)`

Alignment helper macro.

10.20.3.5 `#define MEM_IS_ALIGNED(p) (((size_t)(p) & MEM_ALIGN_MASK) == 0)`

Returns whatever a pointer or memory size is aligned to the type `align_t`.

10.20.4 Typedef Documentation

10.20.4.1 `typedef void*(* memgetfunc_t)(size_t size)`

Memory get function.

Note

This type must be assignment compatible with the `chMemAlloc()` function.

10.21 Heaps

10.21.1 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.

By enabling the `CH_USE_MALLOC_HEAP` option the heap manager will use the runtime-provided `malloc()` and `free()` as back end for the heap APIs instead of the system provided allocator.

Precondition

In order to use the heap APIs the `CH_USE_HEAP` option must be enabled in `chconf.h`.

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 `chHeapInit` (`MemoryHeap` *`heapp`, void *`buf`, `size_t` `size`)
Initializes a memory heap from a static memory area.
- void * `chHeapAlloc` (`MemoryHeap` *`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` (`MemoryHeap` *`heapp`, `size_t` *`sizep`)
Reports the heap status.

10.21.2 Function Documentation

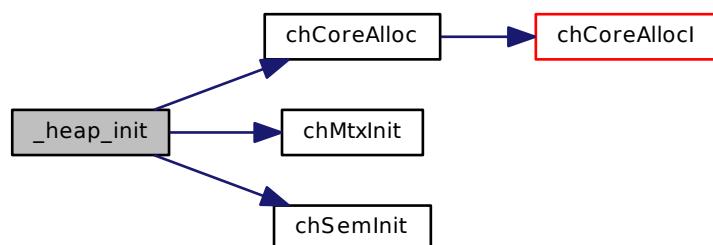
10.21.2.1 void _heap_init(void)

Initializes the default heap.

Function Class:

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

Here is the call graph for this function:



10.21.2.2 void chHeapInit(MemoryHeap * 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.
In order to use this function the option `CH_USE_MALLOC_HEAP` must be disabled.

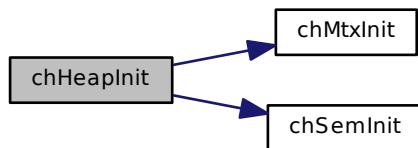
Parameters

out	<code>heapp</code>	pointer to the memory heap descriptor to be initialized
in	<code>buf</code>	heap buffer base
in	<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:



10.21.2.3 void * chHeapAlloc (MemoryHeap * 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

in	<code>heapp</code>	pointer to a heap descriptor or <code>NULL</code> in order to access the default heap.
in	<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

`NULL` 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.

10.21.2.4 void chHeapFree (void * p)

Frees a previously allocated memory block.

Parameters

in *p* 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.

10.21.2.5 size_t chHeapStatus (MemoryHeap * *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.

This function is not implemented when the CH_USE_MALLOC_HEAP configuration option is used (it always returns zero).

Parameters

in *heapp* pointer to a heap descriptor or NULL in order to access the default heap.
in *sizep* 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.

10.22 Memory Pools

10.22.1 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_USE_MEMPOOLS option must be enabled in [chconf.h](#).

Data Structures

- struct [pool_header](#)
Memory pool free object header.
- struct [MemoryPool](#)
Memory pool descriptor.

Functions

- void **chPoolInit** (MemoryPool *mp, size_t size, memgetfunc_t provider)
Initializes an empty memory pool.
- void **chPoolLoadArray** (MemoryPool *mp, void *p, size_t n)
Loads a memory pool with an array of static objects.
- void * **chPoolAlloc** (MemoryPool *mp)
Allocates an object from a memory pool.
- void * **chPoolAlloc** (MemoryPool *mp)
Allocates an object from a memory pool.
- void **chPoolFree** (MemoryPool *mp, void *objp)
Releases an object into a memory pool.
- void **chPoolFree** (MemoryPool *mp, void *objp)
Releases an object into a memory pool.

Macro Functions

- #define **chPoolAdd**(mp, objp) chPoolFree(mp, objp)
Adds an object to a memory pool.
- #define **chPoolAddl**(mp, objp) chPoolFree(mp, objp)
Adds an object to a memory pool.

Defines

- #define **_MEMORYPOOL_DATA**(name, size, provider) {NULL, size, provider}
Data part of a static memory pool initializer.
- #define **MEMORYPOOL_DECL**(name, size, provider) MemoryPool name = **_MEMORYPOOL_DATA**(name, size, provider)
Static memory pool initializer in hungry mode.

10.22.2 Function Documentation

10.22.2.1 void chPoolInit (MemoryPool * mp, size_t size, memgetfunc_t provider)

Initializes an empty memory pool.

Parameters

out	<i>mp</i>	pointer to a MemoryPool 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 NULL 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.

10.22.2.2 void chPoolLoadArray (MemoryPool * 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 MemoryPool 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.

10.22.2.3 void * chPoolAlloc (MemoryPool * 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 MemoryPool structure
----	-----------	---

Returns

The pointer to the allocated object.

Return values

NULL 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:

**10.22.2.4 void * chPoolAlloc (MemoryPool * 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 <code>MemoryPool</code> structure
----	--

Returns

The pointer to the allocated object.

Return values

NULL 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:

**10.22.2.5 void chPoolFree(`MemoryPool` * *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.

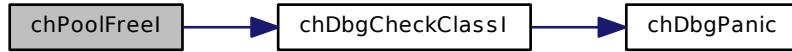
Parameters

in	<i>mp</i> pointer to a <code>MemoryPool</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:



10.22.2.6 void chPoolFree (MemoryPool * 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 MemoryPool 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:



10.22.3 Define Documentation

10.22.3.1 #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

```
10.22.3.2 #define MEMORYPOOL_DECL( name, size, provider ) MemoryPool 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

```
10.22.3.3 #define chPoolAdd( mp, objp ) chPoolFree(mp, objp)
```

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 <code>MemoryPool</code> 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.

```
10.22.3.4 #define chPoolAddl( mp, objp ) chPoolFreel(mp, objp)
```

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 <code>MemoryPool</code> 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.

10.23 Dynamic Threads

10.23.1 Detailed Description

Dynamic threads related APIs and services.

Functions

- `Thread * chThdAddRef (Thread *tp)`
Adds a reference to a thread object.
- `void chThdRelease (Thread *tp)`
Releases a reference to a thread object.
- `Thread * chThdCreateFromHeap (MemoryHeap *heapp, size_t size, tprio_t prio, tfunc_t pf, void *arg)`
Creates a new thread allocating the memory from the heap.
- `Thread * chThdCreateFromMemoryPool (MemoryPool *mp, tprio_t prio, tfunc_t pf, void *arg)`
Creates a new thread allocating the memory from the specified memory pool.

10.23.2 Function Documentation

10.23.2.1 Thread * chThdAddRef (Thread * tp)

Adds a reference to a thread object.

Precondition

The configuration option `CH_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.

10.23.2.2 void chThdRelease (Thread * tp)

Releases a reference to a thread object.

If the references counter reaches zero **and** the thread is in the `THD_STATE_FINAL` state then the thread's memory is returned to the proper allocator.

Precondition

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

Note

Static threads are not affected.

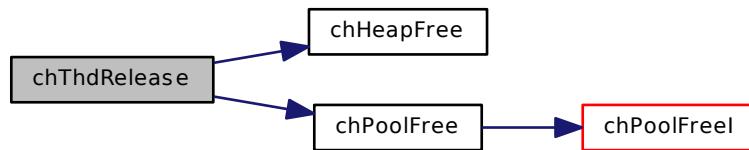
Parameters

in	<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:



10.23.2.3 Thread * chThdCreateFromHeap (MemoryHeap * 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_USE_DYNAMIC and CH_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	<i>heapp</i>	heap from which allocate the memory or NULL for the default heap
in	<i>size</i>	size of the working area to be allocated
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` structure allocated for the thread into the working space area.

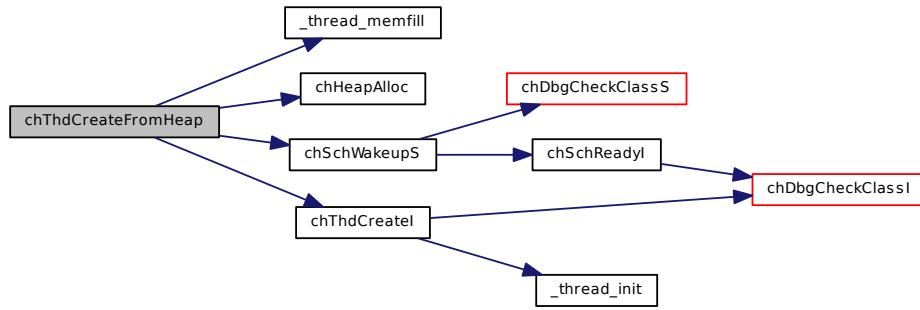
Return values

`NULL` 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:



10.23.2.4 Thread * chThdCreateFromMemoryPool (MemoryPool * 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_USE_DYNAMIC` and `CH_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` structure allocated for the thread into the working space area.

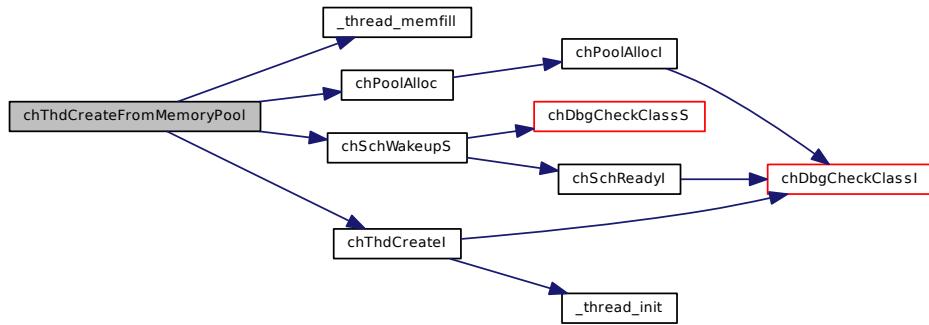
Return values

`NULL` 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:



10.24 Streams and Files

10.24.1 Detailed Description

Stream and Files interfaces.

Modules

- Abstract Sequential Streams
- Abstract File Streams

10.25 Abstract Sequential Streams

10.25.1 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.

Data Structures

- struct `BaseSequentialStreamVMT`
`BaseSequentialStream` virtual methods table.
- struct `BaseSequentialStream`
`Base stream class.`

Macro Functions (`BaseSequentialStream`)

- #define `chSequentialStreamWriter(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.

Defines

- #define _base_sequential_stream_methods
BaseSequentialStream specific methods.
- #define _base_sequential_stream_data
BaseSequentialStream specific data.

10.25.2 Define Documentation

10.25.2.1 #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.

10.25.2.2 #define _base_sequential_stream_data

BaseSequentialStream specific data.

Note

It is empty because *BaseSequentialStream* is only an interface without implementation.

10.25.2.3 #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 <i>BaseSequentialStream</i> 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.

10.25.2.4 #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 BaseSequentialStream 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.

10.25.2.5 #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.

10.25.2.6 #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

Q_RESET 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.

10.26 Abstract File Streams

10.26.1 Detailed Description

This module define an abstract interface for generic data files by extending the [BaseSequentialStream](#) interface. Note that no code is present, data files are just abstract interface-like structures, you should look at the systems as to a set of abstract C++ classes (even if written in C). This system has the advantage to make the access to streams independent from the implementation logic.

The data files interface can be used as base class for high level object types such as an API for a File System implementation.

Data Structures

- struct [BaseFileStreamVMT](#)
BaseFileStream virtual methods table.
- struct [BaseFileStream](#)
Base file stream class.

Macro Functions ([BaseFileStream](#))

- #define [chFileStreamClose\(ip\)](#) ((ip)->vmt->close(ip))
Base file Stream close.
- #define [chFileStreamGetError\(ip\)](#) ((ip)->vmt->geterror(ip))
Returns an implementation dependent error code.
- #define [chFileStreamGetSize\(ip\)](#) ((ip)->vmt->getsize(ip))
Returns the current file size.
- #define [chFileStreamGetPosition\(ip\)](#) ((ip)->vmt->getposition(ip))
Returns the current file pointer position.
- #define [chFileStreamSeek\(ip, offset\)](#) ((ip)->vmt->lseek(ip, offset))
Moves the file current pointer to an absolute position.

Defines

- #define [FILE_OK](#) 0
No error return code.
- #define [FILE_ERROR](#) 0xFFFFFFFFUL
Error code from the file stream methods.
- #define [_base_file_stream_methods](#)
BaseFileStream specific methods.
- #define [_base_file_stream_data](#) [_base_sequential_stream_data](#)
BaseFileStream specific data.

Typedefs

- `typedef uint32_t fileoffset_t`
File offset type.

10.26.2 Define Documentation

10.26.2.1 #define FILE_OK 0

No error return code.

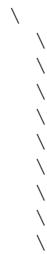
10.26.2.2 #define FILE_ERROR 0xFFFFFFFFUL

Error code from the file stream methods.

10.26.2.3 #define _base_file_stream_methods

Value:

```
_base_sequential_stream_methods
/* File close method.*/
uint32_t (*close)(void *instance);
/* Get last error code method.*/
int (*geterror)(void *instance);
/* File get size method.*/
fileoffset_t (*getsize)(void *instance);
/* File get current position method.*/
fileoffset_t (*getposition)(void *instance);
/* File seek method.*/
uint32_t (*lseek)(void *instance, fileoffset_t offset);
```



`BaseFileStream` specific methods.

10.26.2.4 #define _base_file_stream_data _base_sequential_stream_data

`BaseFileStream` specific data.

Note

It is empty because `BaseFileStream` is only an interface without implementation.

10.26.2.5 #define chFileStreamClose(ip) ((ip)->vmt->close(ip))

Base file Stream close.

The function closes a file stream.

Parameters

in *ip* pointer to a `BaseFileStream` or derived class

Returns

The operation status.

Return values

<code>FILE_OK</code>	no error.
<code>FILE_ERROR</code>	operation failed.

Function Class:

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

10.26.2.6 #define chFileStreamGetError(*ip*) ((*ip*)->vmt->geterror(*ip*))

Returns an implementation dependent error code.

Parameters

in *ip* pointer to a [BaseFileStream](#) or derived class

Returns

Implementation dependent error code.

Function Class:

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

10.26.2.7 #define chFileStreamGetSize(*ip*) ((*ip*)->vmt->getsize(*ip*))

Returns the current file size.

Parameters

in *ip* pointer to a [BaseFileStream](#) or derived class

Returns

The file size.

Function Class:

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

10.26.2.8 #define chFileStreamGetPosition(*ip*) ((*ip*)->vmt->getposition(*ip*))

Returns the current file pointer position.

Parameters

in *ip* pointer to a [BaseFileStream](#) or derived class

Returns

The current position inside the file.

Function Class:

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

10.26.2.9 #define chFileStreamSeek(*ip*, *offset*) ((*ip*)->vmt->lseek(*ip*, *offset*))

Moves the file current pointer to an absolute position.

Parameters

in *ip* pointer to a [BaseFileStream](#) or derived class
in *offset* new absolute position

Returns

The operation status.

Return values

<i>FILE_OK</i>	no error.
<i>FILE_ERROR</i>	operation failed.

Function Class:

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

10.26.3 Typedef Documentation

10.26.3.1 `typedef uint32_t fileoffset_t`

File offset type.

10.27 Registry

10.27.1 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_USE_REGISTRY` option must be enabled in [chconf.h](#).

Data Structures

- struct [chdebug_t](#)
ChibiOS/RT memory signature record.

Functions

- [Thread * chRegFirstThread \(void\)](#)
Returns the first thread in the system.
- [Thread * chRegNextThread \(Thread *tp\)](#)
Returns the thread next to the specified one.

Macro Functions

- #define `chRegSetThreadName(p)` (`currp->p_name = (p)`)
Sets the current thread name.
- #define `chRegGetThreadName(tp)` (`((tp)->p_name)`)
Returns the name of the specified thread.

Defines

- #define `REG_REMOVE(tp)`
Removes a thread from the registry list.
- #define `REG_INSERT(tp)`
Adds a thread to the registry list.

10.27.2 Function Documentation

10.27.2.1 Thread * 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.

10.27.2.2 Thread * chRegNextThread (Thread * 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 *tp* pointer to the thread

Returns

A reference to the next thread.

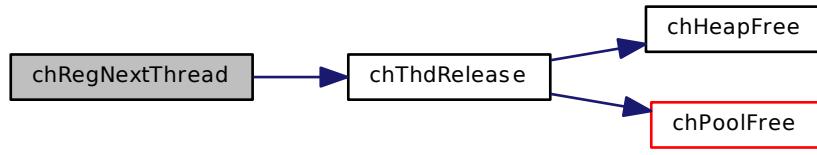
Return values

`NULL` 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:



10.27.3 Define Documentation

10.27.3.1 #define chRegSetThreadName(*p*) (currp->p.name = (*p*))

Sets the current thread name.

Precondition

This function only stores the pointer to the name if the option CH_USE_REGISTRY is enabled else no action is performed.

Parameters

in	<i>p</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.

10.27.3.2 #define chRegGetThreadName(*tp*) ((tp)->p.name)

Returns the name of the specified thread.

Precondition

This function only returns the pointer to the name if the option CH_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

NULL if the thread name has not been set.

10.27.3.3 #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 *tp* thread to remove from the registry

10.27.3.4 #define REG_INSERT(*tp*)

Value:

```
{
  (tp)->p_newer = (Thread *)&rlist;
  (tp)->p_older = rlist.r_older;
  (tp)->p_older->p_newer = rlist.r_older = (tp);
}
```

Adds a thread to the registry list.

Note

This macro is not meant for use in application code.

Parameters

in *tp* thread to add to the registry

10.28 Debug

10.28.1 Detailed Description

Debug APIs and services:

- Runtime system state and call protocol check. The following panic messages can be generated:
 - SV#1, misplaced `chSysDisable()`.
 - SV#2, misplaced `chSysSuspend()`
 - SV#3, misplaced `chSysEnable()`.
 - SV#4, misplaced `chSysLock()`.
 - SV#5, misplaced `chSysUnlock()`.
 - SV#6, misplaced `chSysLockFromIsr()`.
 - SV#7, misplaced `chSysUnlockFromIsr()`.
 - SV#8, misplaced `CH_IRQ_PROLOGUE()`.
 - SV#9, misplaced `CH_IRQ_EPILOGUE()`.
 - SV#10, misplaced I-class function.
 - SV#11, misplaced S-class function.
- 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.

Data Structures

- struct `ch_swc_event_t`
Trace buffer record.
- struct `ch_trace_buffer_t`
Trace buffer header.

Functions

- void `_trace_init` (void)
Trace circular buffer subsystem initialization.
- void `dbg_trace` (Thread *otp)
Inserts in the circular debug trace buffer a context switch record.
- 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 `chDbgPanic` (const char *msg)
Prints a panic message on the console and then halts the system.

Variables

- `cnt_t dbg_isr_cnt`
ISR nesting level.
- `cnt_t dbg_lock_cnt`
Lock nesting level.
- `ch_trace_buffer_t dbg_trace_buffer`
Public trace buffer.
- `const char * dbg_panic_msg`
Pointer to the panic message.

Debug related settings

- `#define CH_TRACE_BUFFER_SIZE 64`
Trace buffer entries.
- `#define CH_STACK_FILL_VALUE 0x55`
Fill value for thread stack area in debug mode.
- `#define CH_THREAD_FILL_VALUE 0xFF`
Fill value for thread area in debug mode.

Macro Functions

- `#define chDbgCheck(c, func)`
Function parameters check.
- `#define chDbgAssert(c, m, r)`
Condition assertion.

10.28.2 Function Documentation

10.28.2.1 void _trace_init(void)

Trace circular buffer subsystem initialization.

Note

Internal use only.

10.28.2.2 void dbg_trace(Thread * otp)

Inserts in the circular debug trace buffer a context switch record.

Parameters

in	<code>otp</code> the thread being switched out
----	--

Function Class:

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

10.28.2.3 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:

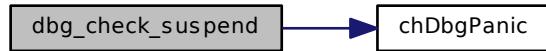
**10.28.2.4 void dbg_check_suspend(void)**

Guard code for [chSysSuspend\(\)](#).

Function Class:

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

Here is the call graph for this function:

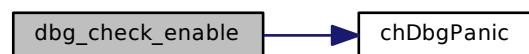
**10.28.2.5 void dbg_check_enable(void)**

Guard code for [chSysEnable\(\)](#).

Function Class:

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

Here is the call graph for this function:



10.28.2.6 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:

**10.28.2.7 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:

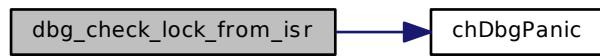
**10.28.2.8 void dbg_check_lock_from_isr(void)**

Guard code for [chSysLockFromIsr\(\)](#).

Function Class:

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

Here is the call graph for this function:



10.28.2.9 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:



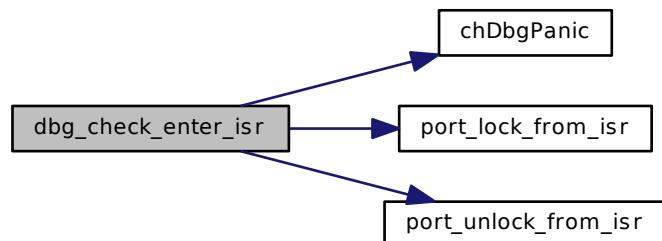
10.28.2.10 void dbg_check_enter_isr(void)

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:



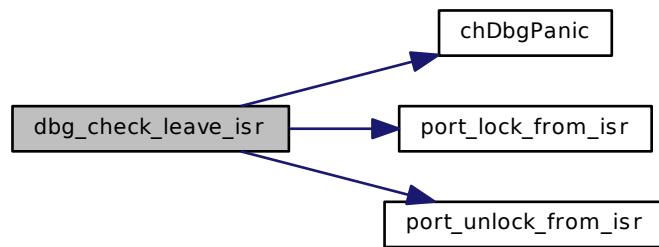
10.28.2.11 void dbg_check_leave_isr(void)

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:

**10.28.2.12 void chDbgCheckClassI(void)**

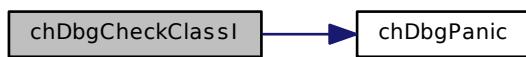
I-class functions context check.

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:

**10.28.2.13 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:

**10.28.2.14 void chDbgPanic (const char * msg)**

Prints a panic message on the console and then halts the system.

Parameters

in *msg* the pointer to the panic message string

10.28.3 Variable Documentation**10.28.3.1 cnt_t dbg_isr_cnt**

ISR nesting level.

10.28.3.2 cnt_t dbg_lock_cnt

Lock nesting level.

10.28.3.3 ch_trace_buffer_t dbg_trace_buffer

Public trace buffer.

10.28.3.4 const char* dbg_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.

10.28.4 Define Documentation**10.28.4.1 #define CH_TRACE_BUFFER_SIZE 64**

Trace buffer entries.

10.28.4.2 #define CH_STACK_FILL_VALUE 0x55

Fill value for thread stack area in debug mode.

10.28.4.3 #define CH_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.

10.28.4.4 #define chDbgCheck(c, func)

Value:

```
{
    if (! (c)) \
        chDbgPanic(__QUOTE_THIS(func) " () ");
}
```

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	<i>c</i>	the condition to be verified to be true
in	<i>func</i>	the undecorated function name

Function Class:

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

10.28.4.5 #define chDbgAssert(c, m, r)

Value:

```
{
    if (! (c)) \
        chDbgPanic(m);
}
```

Condition assertion.

If the condition check fails then the kernel panics with the specified 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 convention for the message is the following:

<function_name>(), #<assert_number>

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>m</i>	the text message
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.

10.29 Internals

10.29.1 Detailed Description

All the functions present in this module, while public, are not OS APIs and should not be directly used in the user applications code.

Data Structures

- struct **ThreadsQueue**
Generic threads bidirectional linked list header and element.
- struct **ThreadsList**
Generic threads single link list, it works like a stack.

Functions

- void **prio_insert** (**Thread** *tp, **ThreadsQueue** *tqp)
Inserts a thread into a priority ordered queue.
- void **queue_insert** (**Thread** *tp, **ThreadsQueue** *tqp)
Inserts a Thread into a queue.
- **Thread** * **fifo_remove** (**ThreadsQueue** *tqp)
Removes the first-out Thread from a queue and returns it.
- **Thread** * **lifo_remove** (**ThreadsQueue** *tqp)
Removes the last-out Thread from a queue and returns it.
- **Thread** * **dequeue** (**Thread** *tp)
Removes a Thread from a queue and returns it.
- void **list_insert** (**Thread** *tp, **ThreadsList** *tlp)
Pushes a Thread on top of a stack list.
- **Thread** * **list_remove** (**ThreadsList** *tlp)
Pops a Thread from the top of a stack list and returns it.

Defines

- #define **queue_init**(tqp) (((tqp)->p_next = (tqp)->p_prev = (Thread *)(tqp));
Threads queue initialization.
- #define **list_init**(tlp) (((tlp)->p_next = (Thread *)(tlp))
Threads list initialization.
- #define **isempty**(p) ((p)->p_next == (Thread *)(p))
Evaluates to TRUE if the specified threads queue or list is empty.
- #define **notempty**(p) ((p)->p_next != (Thread *)(p))
Evaluates to TRUE if the specified threads queue or list is not empty.
- #define **_THREADSQUEUE_DATA**(name) {((Thread *)&name, (Thread *)&name)}
Data part of a static threads queue initializer.
- #define **THREADSQUEUE_DECL**(name) **ThreadsQueue** name = **_THREADSQUEUE_DATA**(name)
Static threads queue initializer.

10.29.2 Function Documentation

10.29.2.1 void prio_insert (Thread * tp, ThreadsQueue * tqp)

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.

10.29.2.2 void queue_insert (Thread * tp, ThreadsQueue * tqp)

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.

10.29.2.3 Thread * fifo_remove (ThreadsQueue * tqp)

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.

10.29.2.4 Thread * lifo_remove (ThreadsQueue * tqp)

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 *tqp* the pointer to the threads list header

Returns

The removed thread pointer.

Function Class:

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

10.29.2.5 Thread * dequeue (Thread * *tp*)

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 *tp* 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.

10.29.2.6 void list_insert (Thread * *tp*, ThreadsList * *tlp*)

Pushes a [Thread](#) on top of a stack list.

Parameters

in *tp* the pointer to the thread to be inserted in the list
in *tlp* the pointer to the threads list header

Function Class:

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

10.29.2.7 Thread * list_remove (ThreadsList * *tlp*)

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 *tlp* the pointer to the threads list header

Returns

The removed thread pointer.

Function Class:

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

10.29.3 Define Documentation

```
10.29.3.1 #define queue_init( tqp ) ((tqp)->p_next = (tqp)->p_prev = (Thread *)(tqp));
```

Threads queue initialization.

Function Class:

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

```
10.29.3.2 #define list_init( tlp ) ((tlp)->p_next = (Thread *)(tlp))
```

Threads list initialization.

Function Class:

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

```
10.29.3.3 #define isempty( p ) ((p)->p_next == (Thread *)(p))
```

Evaluates to TRUE if the specified threads queue or list is empty.

Function Class:

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

```
10.29.3.4 #define notempty( p ) ((p)->p_next != (Thread *)(p))
```

Evaluates to TRUE if the specified threads queue or list is not empty.

Function Class:

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

```
10.29.3.5 #define _THREADSQUEUE_DATA( name ) { (Thread *)&name, (Thread *)&name }
```

Data part of a static threads queue initializer.

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

Parameters

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

```
10.29.3.6 #define THREADSQUEUE_DECL( name ) ThreadsQueue name = _THREADSQUEUE_DATA(name)
```

Static threads queue initializer.

Statically initialized threads queues require no explicit initialization using [queue_init\(\)](#).

Parameters

in *name* the name of the threads queue variable

10.30 Port

10.30.1 Detailed Description

Non portable code templates.

Data Structures

- struct [extctx](#)
Interrupt saved context.
- struct [intctx](#)
System saved context.
- struct [context](#)
Platform dependent part of the [Thread](#) structure.

Functions

- void [port_init](#) (void)
Port-related initialization code.
- void [port_lock](#) (void)
Kernel-lock action.
- void [port_unlock](#) (void)
Kernel-unlock action.
- void [port_lock_from_isr](#) (void)
Kernel-lock action from an interrupt handler.
- void [port_unlock_from_isr](#) (void)
Kernel-unlock action from an interrupt handler.
- void [port_disable](#) (void)
Disables all the interrupt sources.
- void [port_suspend](#) (void)
Disables the interrupt sources below kernel-level priority.
- void [port_enable](#) (void)
Enables all the interrupt sources.
- void [port_wait_for_interrupt](#) (void)
Enters an architecture-dependent IRQ-waiting mode.
- void [port_halt](#) (void)
Halts the system.
- void [port_switch](#) ([Thread](#) *ntp, [Thread](#) *otp)
Performs a context switch between two threads.

Defines

- #define [PORT_IDLE_THREAD_STACK_SIZE](#) 0
Stack size for the system idle thread.
- #define [PORT_INT_REQUIRED_STACK](#) 0
Per-thread stack overhead for interrupts servicing.

- `#define CH_ARCHITECTURE_XXX`
Unique macro for the implemented architecture.
- `#define CH_ARCHITECTURE_NAME ""`
Name of the implemented architecture.
- `#define CH_ARCHITECTURE_VARIANT_NAME ""`
Name of the architecture variant (optional).
- `#define CH_COMPILER_NAME "GCC"`
Name of the compiler supported by this port.
- `#define CH_PORT_INFO ""`
Port-specific information string.
- `#define SETUP_CONTEXT(workspace, wsize, pf, arg)`
Platform dependent part of the `chThdCreateI()` API.
- `#define STACK_ALIGN(n) (((n) - 1) | (sizeof(stkalign_t) - 1)) + 1`
Enforces a correct alignment for a stack area size value.
- `#define THD_WA_SIZE(n)`
Computes the thread working area global size.
- `#define WORKING_AREA(s, n) stkalign_t s[THD_WA_SIZE(n) / sizeof(stkalign_t)]`
Static working area allocation.
- `#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.

Typedefs

- `typedef uint8_t stkalign_t`
Base type for stack and memory alignment.

10.30.2 Function Documentation

10.30.2.1 void port_init(void)

Port-related initialization code.

Note

This function is usually empty.

10.30.2.2 void port_lock(void)

Kernel-lock action.

Usually this function just disables interrupts but may perform more actions.

10.30.2.3 void port_unlock(void)

Kernel-unlock action.

Usually this function just enables interrupts but may perform more actions.

10.30.2.4 void port_lock_from_isr (void)

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.

10.30.2.5 void port_unlock_from_isr (void)

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.

10.30.2.6 void port_disable (void)

Disables all the interrupt sources.

Note

Of course non-maskable interrupt sources are not included.

10.30.2.7 void port_suspend (void)

Disables the interrupt sources below kernel-level priority.

Note

Interrupt sources above kernel level remains enabled.

10.30.2.8 void port_enable (void)

Enables all the interrupt sources.

10.30.2.9 void port_wait_for_interrupt (void)

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.

10.30.2.10 void port_halt (void)

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).

Here is the call graph for this function:



10.30.2.11 void port_switch(Thread * *ntp*, Thread * *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

10.30.3 Define Documentation

10.30.3.1 #define PORT_IDLE_THREAD_STACK_SIZE 0

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.

10.30.3.2 #define PORT_INT_REQUIRED_STACK 0

Per-thread stack overhead for interrupts servicing.

This constant is used in the calculation of the correct working area size. This value can be zero on those architecture where there is a separate interrupt stack and the stack space between intctx and extctx is known to be zero.

10.30.3.3 #define CH_ARCHITECTURE_XXX

Unique macro for the implemented architecture.

10.30.3.4 #define CH_ARCHITECTURE_NAME ""

Name of the implemented architecture.

10.30.3.5 `#define CH_ARCHITECTURE_VARIANT_NAME ""`

Name of the architecture variant (optional).

10.30.3.6 `#define CH_COMPILER_NAME "GCC"`

Name of the compiler supported by this port.

10.30.3.7 `#define CH_PORT_INFO ""`

Port-specific information string.

10.30.3.8 `#define SETUP_CONTEXT(workspace, wsize, pf, arg)`

Value:

```
{                                \
}
```

Platform dependent part of the `chThdCreateI()` API.

This code usually setup the context switching frame represented by an `intctx` structure.

10.30.3.9 `#define STACK_ALIGN(n) (((n) - 1) | (sizeof(stkalign_t) - 1)) + 1`

Enforces a correct alignment for a stack area size value.

10.30.3.10 `#define THD_WA_SIZE(n)`

Value:

```
STACK_ALIGN(sizeof(Thread) +           \
            sizeof(struct intctx) +   \
            sizeof(struct extctx) +   \
            (n) + (PORT_INT_REQUIRED_STACK)) \
```

Computes the thread working area global size.

10.30.3.11 `#define WORKING_AREA(s, n) stkalign_t s[THD_WA_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.

10.30.3.12 `#define PORT_IRQ_PROLOGUE()`

IRQ prologue code.

This macro must be inserted at the start of all IRQ handlers enabled to invoke system APIs.

10.30.3.13 `#define PORT_IRQ_EPILOGUE()`

IRQ epilogue code.

This macro must be inserted at the end of all IRQ handlers enabled to invoke system APIs.

10.30.3.14 #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.

10.30.3.15 #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.

Not all architectures support fast interrupts, in this case this macro must be omitted.

10.30.4 Typedef Documentation

10.30.4.1 typedef uint8_t stkalign_t

Base type for stack and memory alignment.

10.31 ADC Driver

10.31.1 Detailed Description

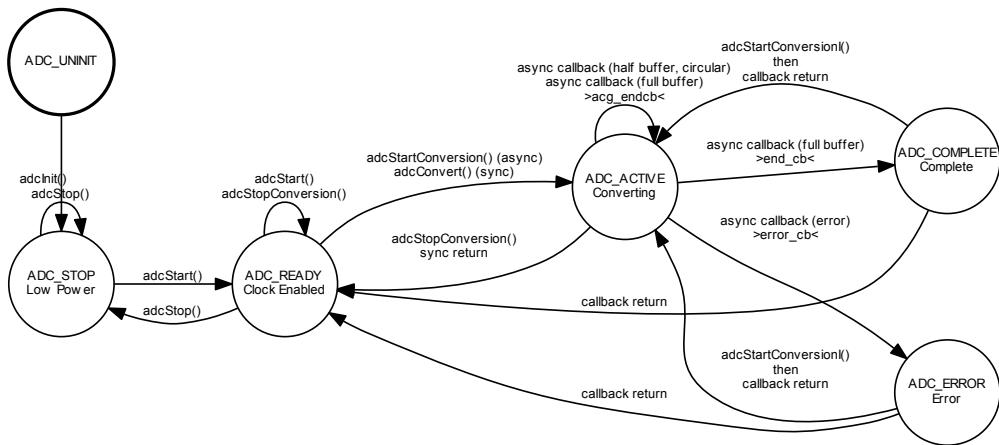
Generic ADC Driver. This module implements a generic ADC (Analog to Digital Converter) driver supporting a variety of buffer and conversion modes.

Precondition

In order to use the ADC driver the `HAL_USE_ADC` option must be enabled in `halconf.h`.

10.31.2 Driver State Machine

The driver implements a state machine internally, not all the driver functionalities can be used in any moment, any transition not explicitly shown in the following diagram has to be considered an error and shall be captured by an assertion (if enabled).



10.31.3 ADC Operations

The ADC driver is quite complex, an explanation of the terminology and of the operational details follows.

10.31.3.1 ADC Conversion Groups

The [ADCConversionGroup](#) is the objects that specifies a physical conversion operation. This structure contains some standard fields and several implementation-dependent fields.

The standard fields define the CG mode, the number of channels belonging to the CG and the optional callbacks.

The implementation-dependent fields specify the physical ADC operation mode, the analog channels belonging to the group and any other implementation-specific setting. Usually the extra fields just mirror the physical ADC registers, please refer to the vendor's MCU Reference Manual for details about the available settings. Details are also available into the documentation of the ADC low level drivers and in the various sample applications.

10.31.3.2 ADC Conversion Modes

The driver supports several conversion modes:

- **One Shot**, the driver performs a single group conversion then stops.
- **Linear Buffer**, the driver performs a series of group conversions then stops. This mode is like a one shot conversion repeated N times, the buffer pointer increases after each conversion. The buffer is organized as an S(CG)*N samples matrix, when S(CG) is the conversion group size (number of channels) and N is the buffer depth (number of repeated conversions).
- **Circular Buffer**, much like the linear mode but the operation does not stop when the buffer is filled, it is automatically restarted with the buffer pointer wrapping back to the buffer base.

10.31.3.3 ADC Callbacks

The driver is able to invoke callbacks during the conversion process. A callback is invoked when the operation has been completed or, in circular mode, when the buffer has been filled and the operation is restarted. In circular mode a callback is also invoked when the buffer is half filled.

The "half filled" and "filled" callbacks in circular mode allow to implement "streaming processing" of the sampled data, while the driver is busy filling one half of the buffer the application can process the other half, this allows for continuous interleaved operations.

The driver is not thread safe for performance reasons, if you need to access the ADC bus from multiple threads then use the `adcAcquireBus()` and `adcReleaseBus()` APIs in order to gain exclusive access.

Data Structures

- struct `ADCConversionGroup`
Conversion group configuration structure.
- struct `ADCConfig`
Driver configuration structure.
- struct `ADCDriver`
Structure representing an ADC driver.

Functions

- void `adclInit (void)`
ADC Driver initialization.
- void `adcObjectInit (ADCDriver *adcp)`
Initializes the standard part of a `ADCDriver` structure.
- void `adcStart (ADCDriver *adcp, const ADCConfig *config)`
Configures and activates the ADC peripheral.
- void `adcStop (ADCDriver *adcp)`
Deactivates the ADC peripheral.
- void `adcStartConversion (ADCDriver *adcp, const ADCConversionGroup *grpp, adcsample_t *samples, size_t depth)`
Starts an ADC conversion.
- void `adcStartConversionI (ADCDriver *adcp, const ADCConversionGroup *grpp, adcsample_t *samples, size_t depth)`
Starts an ADC conversion.
- void `adcStopConversion (ADCDriver *adcp)`
Stops an ongoing conversion.
- void `adcStopConversionI (ADCDriver *adcp)`
Stops an ongoing conversion.
- void `adcAcquireBus (ADCDriver *adcp)`
Gains exclusive access to the ADC peripheral.
- void `adcReleaseBus (ADCDriver *adcp)`
Releases exclusive access to the ADC peripheral.
- msg_t `adcConvert (ADCDriver *adcp, const ADCConversionGroup *grpp, adcsample_t *samples, size_t depth)`
Performs an ADC conversion.
- void `adc_lld_init (void)`
Low level ADC driver initialization.
- void `adc_lld_start (ADCDriver *adcp)`
Configures and activates the ADC peripheral.
- void `adc_lld_stop (ADCDriver *adcp)`
Deactivates the ADC peripheral.
- void `adc_lld_start_conversion (ADCDriver *adcp)`
Starts an ADC conversion.
- void `adc_lld_stop_conversion (ADCDriver *adcp)`
Stops an ongoing conversion.

Variables

- **ADCDriver ADCT1**
ADC1 driver identifier.

ADC configuration options

- **#define ADC_USE_WAIT TRUE**
Enables synchronous APIs.
- **#define ADC_USE_MUTUAL_EXCLUSION TRUE**
Enables the `adcAcquireBus()` and `adcReleaseBus()` APIs.

Low Level driver helper macros

- **#define _adc_reset_i(adcp)**
Resumes a thread waiting for a conversion completion.
- **#define _adc_reset_s(adcp)**
Resumes a thread waiting for a conversion completion.
- **#define _adc_wakeup_isr(adcp)**
Wakes up the waiting thread.
- **#define _adc_timeout_isr(adcp)**
Wakes up the waiting thread with a timeout message.
- **#define _adc_isr_half_code(adcp)**
Common ISR code, half buffer event.
- **#define _adc_isr_full_code(adcp)**
Common ISR code, full buffer event.
- **#define _adc_isr_error_code(adcp, err)**
Common ISR code, error event.

Configuration options

- **#define PLATFORM_ADC_USE_ADC1 FALSE**
ADC1 driver enable switch.

Typedefs

- **typedef uint16_t adcsample_t**
ADC sample data type.
- **typedef uint16_t adc_channels_num_t**
Channels number in a conversion group.
- **typedef struct ADCDriver ADCDriver**
Type of a structure representing an ADC driver.
- **typedef void(* adccallback_t)(ADCDriver *adcp, adcsample_t *buffer, size_t n)**
ADC notification callback type.
- **typedef void(* adcerrorcallback_t)(ADCDriver *adcp, adcerror_t err)**
ADC error callback type.

Enumerations

- enum `adcstate_t` {

`ADC_UNINIT` = 0, `ADC_STOP` = 1, `ADC_READY` = 2, `ADC_ACTIVE` = 3,

`ADC_COMPLETE` = 4, `ADC_ERROR` = 5 }

Driver state machine possible states.
- enum `adcerror_t` { `ADC_ERR_DMAFAILURE` = 0, `ADC_ERR_OVERFLOW` = 1 }

Possible ADC failure causes.

10.31.4 Function Documentation

10.31.4.1 void adcInit(void)

ADC Driver initialization.

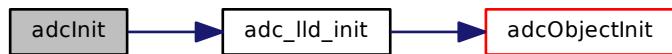
Note

This function is implicitly invoked by `halInit()`, there is no need to explicitly initialize the driver.

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:



10.31.4.2 void adcObjectInit(ADCDriver * adcp)

Initializes the standard part of a `ADCDriver` structure.

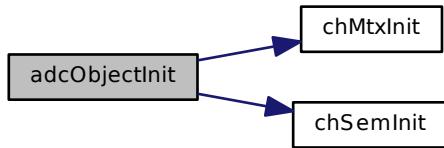
Parameters

`out adcp` pointer to the `ADCDriver` 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:



10.31.4.3 void adcStart (ADCDriver * *adcp*, const ADCConfig * *config*)

Configures and activates the ADC peripheral.

Parameters

in	<i>adcp</i>	pointer to the ADCDriver object
in	<i>config</i>	pointer to the ADCConfig object. Depending on the implementation the value can be NULL.

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:



10.31.4.4 void adcStop (ADCDriver * *adcp*)

Deactivates the ADC peripheral.

Parameters

in	<i>adcp</i>	pointer to the ADCDriver 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:



10.31.4.5 void adcStartConversion (ADCDriver * *adcp*, const ADCConversionGroup * *grpp*, adcsample_t * *samples*, size_t *depth*)

Starts an ADC conversion.

Starts an asynchronous conversion operation.

Note

The buffer is organized as a matrix of M*N elements where M is the channels number configured into the conversion group and N is the buffer depth. The samples are sequentially written into the buffer with no gaps.

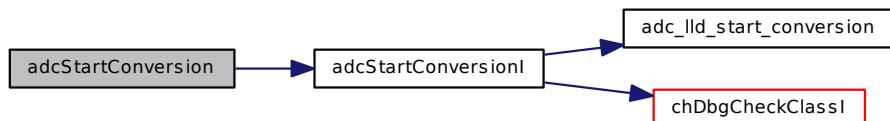
Parameters

in	<i>adcp</i>	pointer to the ADCDriver object
in	<i>grpp</i>	pointer to a ADCConversionGroup object
out	<i>samples</i>	pointer to the samples buffer
in	<i>depth</i>	buffer depth (matrix rows number). The buffer depth must be one or an even number.

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:



10.31.4.6 void adcStartConversionl (ADCDriver * *adcp*, const ADCConversionGroup * *grpp*, adcsample_t * *samples*, size_t *depth*)

Starts an ADC conversion.

Starts an asynchronous conversion operation.

Postcondition

The callbacks associated to the conversion group will be invoked on buffer fill and error events.

Note

The buffer is organized as a matrix of M*N elements where M is the channels number configured into the conversion group and N is the buffer depth. The samples are sequentially written into the buffer with no gaps.

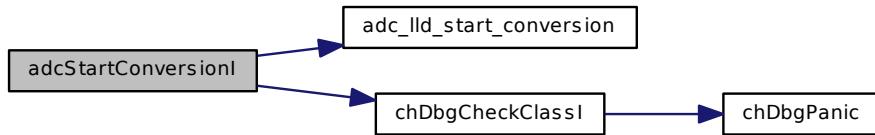
Parameters

in	<i>adcp</i>	pointer to the ADCDriver object
in	<i>grpp</i>	pointer to a ADCConversionGroup object
out	<i>samples</i>	pointer to the samples buffer
in	<i>depth</i>	buffer depth (matrix rows number). The buffer depth must be one or an even number.

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:

**10.31.4.7 void adcStopConversion (ADCDriver * *adcp*)**

Stops an ongoing conversion.

This function stops the currently ongoing conversion and returns the driver in the `ADC_READY` state. If there was no conversion being processed then the function does nothing.

Parameters

in	<i>adcp</i>	pointer to the ADCDriver 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:



10.31.4.8 void adcStopConversionl (ADCDriver * *adcp*)

Stops an ongoing conversion.

This function stops the currently ongoing conversion and returns the driver in the `ADC_READY` state. If there was no conversion being processed then the function does nothing.

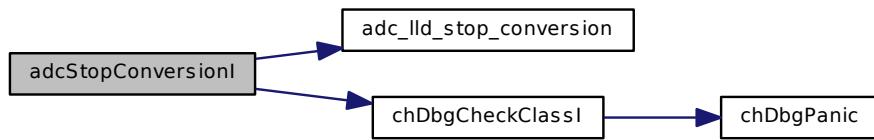
Parameters

in *adcp* pointer to the `ADCDriver` object

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:



10.31.4.9 void adcAcquireBus (ADCDriver * *adcp*)

Gains exclusive access to the ADC peripheral.

This function tries to gain ownership to the ADC bus, if the bus is already being used then the invoking thread is queued.

Precondition

In order to use this function the option `ADC_USE_MUTUAL_EXCLUSION` must be enabled.

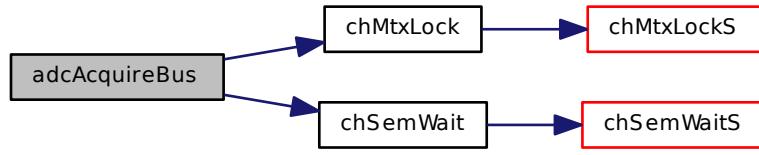
Parameters

in *adcp* pointer to the `ADCDriver` 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:



10.31.4.10 void adcReleaseBus (ADCDriver * adcp)

Releases exclusive access to the ADC peripheral.

Precondition

In order to use this function the option `ADC_USE_MUTUAL_EXCLUSION` must be enabled.

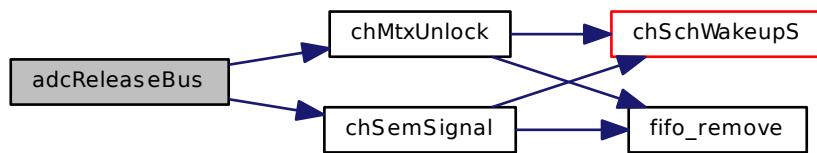
Parameters

in `adcp` pointer to the `ADCDriver` 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:



10.31.4.11 msg_t adcConvert (ADCDriver * adcp, const ADCConversionGroup * grpp, adcsample_t * samples, size_t depth)

Performs an ADC conversion.

Performs a synchronous conversion operation.

Note

The buffer is organized as a matrix of M*N elements where M is the channels number configured into the conversion group and N is the buffer depth. The samples are sequentially written into the buffer with no gaps.

Parameters

in	<i>adcp</i>	pointer to the <code>ADCDriver</code> object
in	<i>grpp</i>	pointer to a <code>ADCConversionGroup</code> object
out	<i>samples</i>	pointer to the samples buffer
in	<i>depth</i>	buffer depth (matrix rows number). The buffer depth must be one or an even number.

Returns

The operation result.

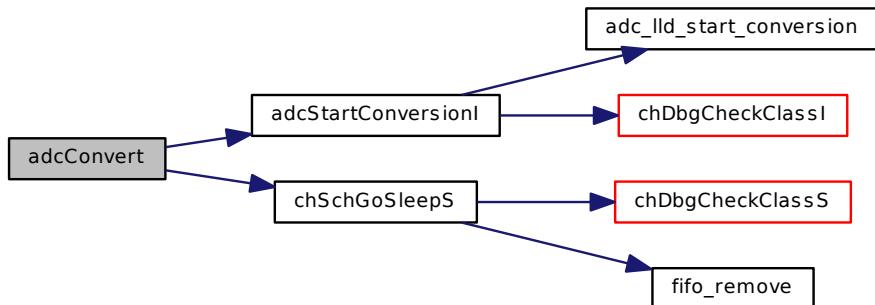
Return values

<i>RDY_OK</i>	Conversion finished.
<i>RDY_RESET</i>	The conversion has been stopped using <code>acdStopConversion()</code> or <code>acdStopConversionI()</code> , the result buffer may contain incorrect data.
<i>RDY_TIMEOUT</i>	The conversion has been stopped because an hardware error.

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:

**10.31.4.12 void adc_lld_init(void)**

Low level ADC driver initialization.

Function Class:

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

Here is the call graph for this function:



10.31.4.13 void adc_lld_start (ADCDriver * *adcp*)

Configures and activates the ADC peripheral.

Parameters

in *adcp* pointer to the `ADCDriver` object

Function Class:

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

10.31.4.14 void adc_lld_stop (ADCDriver * *adcp*)

Deactivates the ADC peripheral.

Parameters

in *adcp* pointer to the `ADCDriver` object

Function Class:

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

10.31.4.15 void adc_lld_start_conversion (ADCDriver * *adcp*)

Starts an ADC conversion.

Parameters

in *adcp* pointer to the `ADCDriver` object

Function Class:

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

10.31.4.16 void adc_lld_stop_conversion (ADCDriver * *adcp*)

Stops an ongoing conversion.

Parameters

in *adcp* pointer to the [ADCDriver](#) object

Function Class:

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

10.31.5 Variable Documentation**10.31.5.1 ADCDriver ADCD1**

ADC1 driver identifier.

10.31.6 Define Documentation**10.31.6.1 #define ADC_USE_WAIT TRUE**

Enables synchronous APIs.

Note

Disabling this option saves both code and data space.

10.31.6.2 #define ADC_USE_MUTUAL_EXCLUSION TRUE

Enables the [adcAcquireBus\(\)](#) and [adcReleaseBus\(\)](#) APIs.

Note

Disabling this option saves both code and data space.

10.31.6.3 #define _adc_reset_i(*adcp*)**Value:**

```
{
  if ((adcp)->thread != NULL) {
    Thread *tp = (adcp)->thread;
    (adcp)->thread = NULL;
    tp->p_u.rdymsg = RDY_RESET;
    chSchReadyI(tp);
  }
}
```

Resumes a thread waiting for a conversion completion.

Parameters

in *adcp* pointer to the [ADCDriver](#) object

Function Class:

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

10.31.6.4 #define _adc_reset_s(*adcp*)**Value:**

```
{
    if ((adcp)->thread != NULL) {
        Thread *tp = (adcp)->thread;
        (adcp)->thread = NULL;
        chSchWakeupS(tp, RDY_RESET);
    }
}
```

Resumes a thread waiting for a conversion completion.

Parameters

in *adcp* pointer to the [ADCDriver](#) object

Function Class:

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

10.31.6.5 #define _adc_wakeup_isr(*adcp*)

Value:

```
{
    chSysLockFromIsr();
    if ((adcp)->thread != NULL) {
        Thread *tp;
        tp = (adcp)->thread;
        (adcp)->thread = NULL;
        tp->p_u.rdymsg = RDY_OK;
        chSchReadyI(tp);
    }
    chSysUnlockFromIsr();
}
```

Wakes up the waiting thread.

Parameters

in *adcp* pointer to the [ADCDriver](#) object

Function Class:

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

10.31.6.6 #define _adc_timeout_isr(*adcp*)

Value:

```
{
    chSysLockFromIsr();
    if ((adcp)->thread != NULL) {
        Thread *tp;
        tp = (adcp)->thread;
        (adcp)->thread = NULL;
        tp->p_u.rdymsg = RDY_TIMEOUT;
        chSchReadyI(tp);
    }
    chSysUnlockFromIsr();
}
```

Wakes up the waiting thread with a timeout message.

Parameters

in *adcp* pointer to the `ADCDriver` object

Function Class:

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

10.31.6.7 `#define _adc_isr_half_code(adc)`

Value:

```
{           \
    if ((adcp)->grpp->end_cb != NULL) { \
        (adcp)->grpp->end_cb(adcp, (adcp)->samples, (adcp)->depth / 2); \
    }           \
}
```

Common ISR code, half buffer event.

This code handles the portable part of the ISR code:

- Callback invocation.

Note

This macro is meant to be used in the low level drivers implementation only.

Parameters

in *adcp* pointer to the `ADCDriver` object

Function Class:

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

10.31.6.8 `#define _adc_isr_full_code(adc)`

Common ISR code, full buffer event.

This code handles the portable part of the ISR code:

- Callback invocation.
- Waiting thread wakeup, if any.
- Driver state transitions.

Note

This macro is meant to be used in the low level drivers implementation only.

Parameters

in *adcp* pointer to the `ADCDriver` object

Function Class:

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

10.31.6.9 #define _adc_isr_error_code(*adcp*, *err*)

Value:

```
{
    adc_lld_stop_conversion(adcp);
    if ((adcp)>grpp->error_cb != NULL) {
        (adcp)>state = ADC_ERROR;
        (adcp)>grpp->error_cb(adcp, err);
        if ((adcp)>state == ADC_ERROR)
            (adcp)>state = ADC_READY;
    }
    (adcp)>grpp = NULL;
    _adc_timeout_isr(adcp);
}
```

Common ISR code, error event.

This code handles the portable part of the ISR code:

- Callback invocation.
- Waiting thread timeout signaling, if any.
- Driver state transitions.

Note

This macro is meant to be used in the low level drivers implementation only.

Parameters

in	<i>adcp</i>	pointer to the ADCDriver object
in	<i>err</i>	platform dependent error code

Function Class:

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

10.31.6.10 #define PLATFORM_ADC_USE_ADC1 FALSE

ADC1 driver enable switch.

If set to TRUE the support for ADC1 is included.

Note

The default is FALSE.

10.31.7 Typedef Documentation

10.31.7.1 **typedef uint16_t adcsample_t**

ADC sample data type.

10.31.7.2 **typedef uint16_t adc_channels_num_t**

Channels number in a conversion group.

10.31.7.3 **typedef struct ADCCDriver ADCCDriver**

Type of a structure representing an ADC driver.

10.31.7.4 `typedef void(* adccallback_t)(ADCDriver *adcp, adcsample_t *buffer, size_t n)`

ADC notification callback type.

Parameters

in	<code>adcp</code>	pointer to the <code>ADCDriver</code> object triggering the callback
in	<code>buffer</code>	pointer to the most recent samples data
in	<code>n</code>	number of buffer rows available starting from <code>buffer</code>

10.31.7.5 `typedef void(* adcerrorcallback_t)(ADCDriver *adcp, adcerror_t err)`

ADC error callback type.

Parameters

in	<code>adcp</code>	pointer to the <code>ADCDriver</code> object triggering the callback
in	<code>err</code>	ADC error code

10.31.8 Enumeration Type Documentation

10.31.8.1 `enum adcstate_t`

Driver state machine possible states.

Enumerator:

- `ADC_UNINIT`** Not initialized.
- `ADC_STOP`** Stopped.
- `ADC_READY`** Ready.
- `ADC_ACTIVE`** Converting.
- `ADC_COMPLETE`** Conversion complete.
- `ADC_ERROR`** Conversion complete.

10.31.8.2 `enum adcerror_t`

Possible ADC failure causes.

Note

Error codes are architecture dependent and should not relied upon.

Enumerator:

- `ADC_ERR_DMAFAILURE`** DMA operations failure.
- `ADC_ERR_OVERFLOW`** ADC overflow condition.

10.32 CAN Driver

10.32.1 Detailed Description

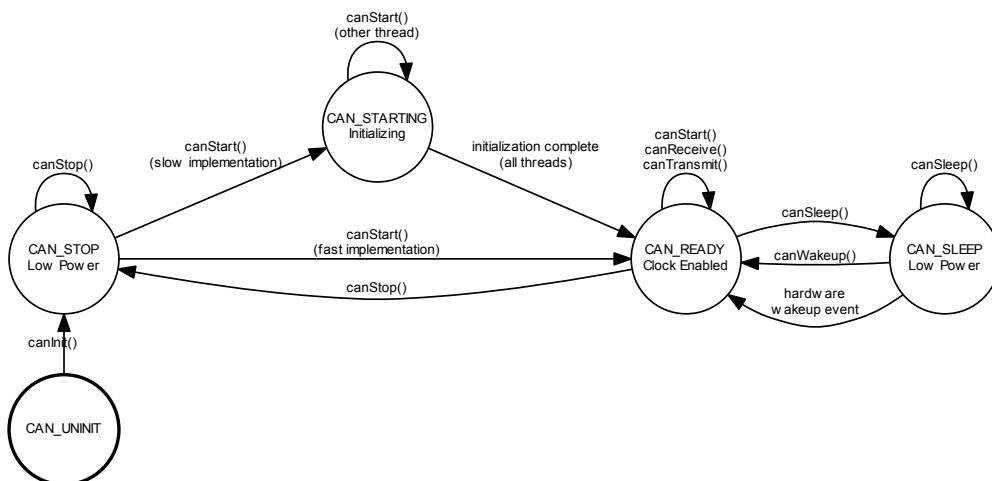
Generic CAN Driver. This module implements a generic CAN (Controller Area Network) driver allowing the exchange of information at frame level.

Precondition

In order to use the CAN driver the `HAL_USE_CAN` option must be enabled in `halconf.h`.

10.32.2 Driver State Machine

The driver implements a state machine internally, not all the driver functionalities can be used in any moment, any transition not explicitly shown in the following diagram has to be considered an error and shall be captured by an assertion (if enabled).



Data Structures

- struct `CANTxFrame`
CAN transmission frame.
- struct `CANRxFrame`
CAN received frame.
- struct `CANFilter`
CAN filter.
- struct `CANConfig`
Driver configuration structure.
- struct `CANDriver`
Structure representing an CAN driver.

Functions

- void `canInit` (void)
CAN Driver initialization.
- void `canObjectInit` (`CANDriver` *`cancp`)
Initializes the standard part of a `CANDriver` structure.
- void `canStart` (`CANDriver` *`cancp`, const `CANConfig` *`config`)
Configures and activates the CAN peripheral.

- void `canStop (CANDriver *canp)`
Deactivates the CAN peripheral.
- `msg_t canTransmit (CANDriver *canp, canmbx_t mailbox, const CANTxFrame *ctfp, systime_t timeout)`
Can frame transmission.
- `msg_t canReceive (CANDriver *canp, canmbx_t mailbox, CANRxFrame *crfp, systime_t timeout)`
Can frame receive.
- void `canSleep (CANDriver *canp)`
Enters the sleep mode.
- void `canWakeup (CANDriver *canp)`
Forces leaving the sleep mode.
- void `can_lld_init (void)`
Low level CAN driver initialization.
- void `can_lld_start (CANDriver *canp)`
Configures and activates the CAN peripheral.
- void `can_lld_stop (CANDriver *canp)`
Deactivates the CAN peripheral.
- `bool_t can_lld_is_tx_empty (CANDriver *canp, canmbx_t mailbox)`
Determines whether a frame can be transmitted.
- void `can_lld_transmit (CANDriver *canp, canmbx_t mailbox, const CANTxFrame *ctfp)`
Inserts a frame into the transmit queue.
- `bool_t can_lld_is_rx_nonempty (CANDriver *canp, canmbx_t mailbox)`
Determines whether a frame has been received.
- void `can_lld_receive (CANDriver *canp, canmbx_t mailbox, CANRxFrame *crfp)`
Receives a frame from the input queue.
- void `can_lld_sleep (CANDriver *canp)`
Enters the sleep mode.
- void `can_lld_wakeup (CANDriver *canp)`
Forces leaving the sleep mode.

Variables

- `CANDriver CAND1`
CAN1 driver identifier.

CAN status flags

- `#define CAN_LIMIT_WARNING 1`
Errors rate warning.
- `#define CAN_LIMIT_ERROR 2`
Errors rate error.
- `#define CAN_BUS_OFF_ERROR 4`
Bus off condition reached.
- `#define CAN_FRAMING_ERROR 8`
Framing error of some kind on the CAN bus.
- `#define CAN_OVERFLOW_ERROR 16`
Overflow in receive queue.

CAN configuration options

- `#define CAN_USE_SLEEP_MODE TRUE`
Sleep mode related APIs inclusion switch.

Macro Functions

- `#define CAN_MAILBOX_TO_MASK(mbx) (1 << ((mbx) - 1))`
Converts a mailbox index to a bit mask.

Configuration options

- `#define PLATFORM_CAN_USE_CAN1 FALSE`
CAN1 driver enable switch.

Defines

- `#define CAN_ANY_MAILBOX 0`
Special mailbox identifier.
- `#define CAN_SUPPORTS_SLEEP TRUE`
This switch defines whether the driver implementation supports a low power switch mode with automatic an wakeup feature.
- `#define CAN_TX_MAILBOXES 3`
This implementation supports three transmit mailboxes.
- `#define CAN_RX_MAILBOXES 2`
This implementation supports two receive mailboxes.

Typedefs

- `typedef uint32_t canmbx_t`
Type of a transmission mailbox index.

Enumerations

- `enum canstate_t {`
`CAN_UNINIT = 0, CAN_STOP = 1, CAN_STARTING = 2, CAN_READY = 3,`
`CAN_SLEEP = 4 }`
Driver state machine possible states.

10.32.3 Function Documentation

10.32.3.1 void caninit(void)

CAN Driver initialization.

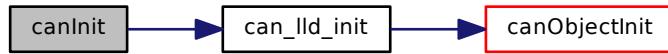
Note

This function is implicitly invoked by `halInit()`, there is no need to explicitly initialize the driver.

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:



10.32.3.2 void canObjectInit (**CANDriver** * *canp*)

Initializes the standard part of a [CANDriver](#) structure.

Parameters

out	<i>canp</i> pointer to the CANDriver 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:



10.32.3.3 void canStart (**CANDriver** * *canp*, const **CANConfig** * *config*)

Configures and activates the CAN peripheral.

Note

Activating the CAN bus can be a slow operation this this function is not atomic, it waits internally for the initialization to complete.

Parameters

in	<i>canp</i> pointer to the CANDriver object
in	<i>config</i> pointer to the CANConfig object. Depending on the implementation the value can be NULL.

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:



10.32.3.4 void canStop (CANDriver * *canp*)

Deactivates the CAN peripheral.

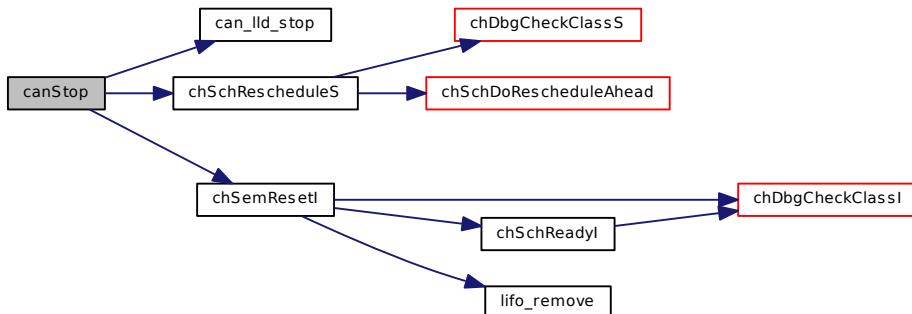
Parameters

in	<i>canp</i> pointer to the CANDriver 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:



10.32.3.5 msg_t canTransmit (CANDriver * *canp*, canmbx_t *mailbox*, const CANTxFr ame * *ctfp*, systime_t *timeout*)

Can frame transmission.

The specified frame is queued for transmission, if the hardware queue is full then the invoking thread is queued.

Note

Trying to transmit while in sleep mode simply enqueues the thread.

Parameters

in	<i>canp</i> pointer to the CANDriver object
in	<i>mailbox</i> mailbox number, CAN_ANY_MAILBOX for any mailbox

in *ctfp* pointer to the CAN frame to be transmitted
 in *timeout* the number of ticks before the operation timeouts, the following special values are allowed:
 • *TIME_IMMEDIATE* immediate timeout.
 • *TIME_INFINITE* no timeout.

Returns

The operation result.

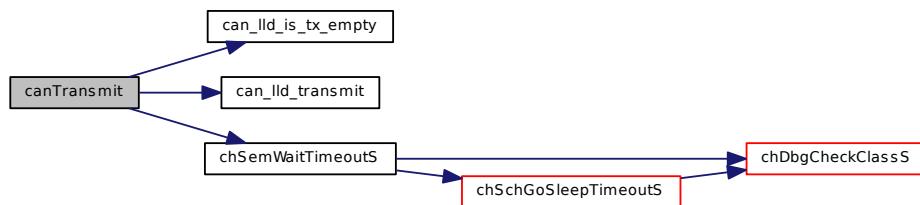
Return values

RDY_OK the frame has been queued for transmission.
RDY_TIMEOUT The operation has timed out.
RDY_RESET The driver has been stopped while waiting.

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:

**10.32.3.6 msg_t canReceive (CANDriver * *canp*, canmbx_t *mailbox*, CANRxFrame * *crfp*, systime_t *timeout*)**

Can frame receive.

The function waits until a frame is received.

Note

Trying to receive while in sleep mode simply enqueues the thread.

Parameters

in *canp* pointer to the [CANDriver](#) object
 in *mailbox* mailbox number, [CAN_ANY_MAILBOX](#) for any mailbox
 out *crfp* pointer to the buffer where the CAN frame is copied
 in *timeout* the number of ticks before the operation timeouts, the following special values are allowed:
 • *TIME_IMMEDIATE* immediate timeout (useful in an event driven scenario where a thread never blocks for I/O).
 • *TIME_INFINITE* no timeout.

Returns

The operation result.

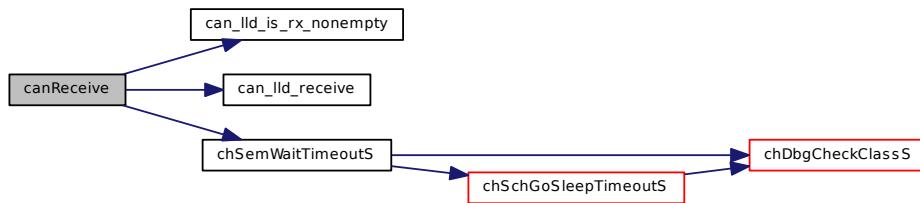
Return values

- RDY_OK* a frame has been received and placed in the buffer.
- RDY_TIMEOUT* The operation has timed out.
- RDY_RESET* The driver has been stopped while waiting.

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:

**10.32.3.7 void canSleep (CANDriver * canp)**

Enters the sleep mode.

This function puts the CAN driver in sleep mode and broadcasts the `sleep_event` event source.

Precondition

In order to use this function the option `CAN_USE_SLEEP_MODE` must be enabled and the `CAN_SUPPORTS_SLEEP` mode must be supported by the low level driver.

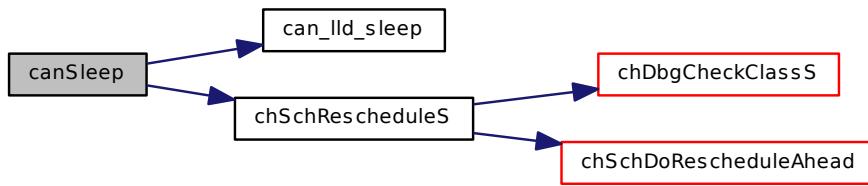
Parameters

in *canp* pointer to the `CANDriver` 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:



10.32.3.8 void canWakeup (CANDriver * *canp*)

Enforces leaving the sleep mode.

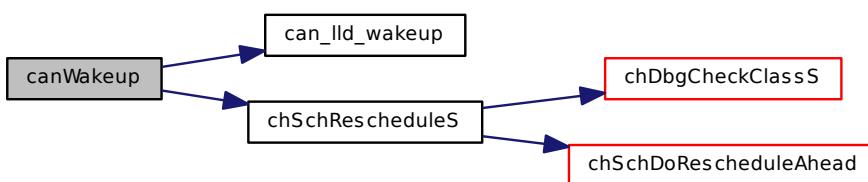
Note

The sleep mode is supposed to be usually exited automatically by an hardware event.

Parameters

in *canp* pointer to the `CANDriver` object

Here is the call graph for this function:



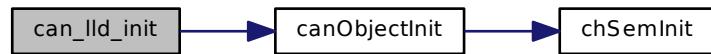
10.32.3.9 void can_lld_init (void)

Low level CAN driver initialization.

Function Class:

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

Here is the call graph for this function:



10.32.3.10 void can_lld_start (**CANDriver** * *canp*)

Configures and activates the CAN peripheral.

Parameters

in	<i>canp</i> pointer to the CANDriver object
----	--

Function Class:

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

10.32.3.11 void can_lld_stop (**CANDriver** * *canp*)

Deactivates the CAN peripheral.

Parameters

in	<i>canp</i> pointer to the CANDriver object
----	--

Function Class:

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

10.32.3.12 **bool_t** can_lld_is_tx_empty (**CANDriver** * *canp*, **canmbx_t** *mailbox*)

Determines whether a frame can be transmitted.

Parameters

in	<i>canp</i> pointer to the CANDriver object
in	<i>mailbox</i> mailbox number, CAN_ANY_MAILBOX for any mailbox

Returns

The queue space availability.

Return values

<i>FALSE</i>	no space in the transmit queue.
<i>TRUE</i>	transmit slot available.

Function Class:

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

10.32.3.13 void can_lld_transmit (**CANDriver** * *canp*, **canmbx_t** *mailbox*, const **CANTxFrame** * *ctfp*)

Inserts a frame into the transmit queue.

Parameters

in	<i>canp</i>	pointer to the CANDriver object
in	<i>ctfp</i>	pointer to the CAN frame to be transmitted
in	<i>mailbox</i>	mailbox number, CAN_ANY_MAILBOX for any mailbox

Function Class:

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

10.32.3.14 bool_t can_lld_is_rx_nonempty (**CANDriver** * *canp*, **canmbx_t** *mailbox*)

Determines whether a frame has been received.

Parameters

in	<i>canp</i>	pointer to the CANDriver object
in	<i>mailbox</i>	mailbox number, CAN_ANY_MAILBOX for any mailbox

Returns

The queue space availability.

Return values

<i>FALSE</i>	no space in the transmit queue.
<i>TRUE</i>	transmit slot available.

Function Class:

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

10.32.3.15 void can_lld_receive (**CANDriver** * *canp*, **canmbx_t** *mailbox*, **CANRxFrame** * *crfp*)

Receives a frame from the input queue.

Parameters

in	<i>canp</i>	pointer to the CANDriver object
in	<i>mailbox</i>	mailbox number, CAN_ANY_MAILBOX for any mailbox
out	<i>crfp</i>	pointer to the buffer where the CAN frame is copied

Function Class:

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

10.32.3.16 void can_lld_sleep (**CANDriver** * *canp*)

Enters the sleep mode.

Parameters

in	<i>canp</i>	pointer to the CANDriver object
----	-------------	--

Function Class:

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

10.32.3.17 void can_lld_wakeup (**CANDriver** * *canp*)

Enforces leaving the sleep mode.

Parameters

in *canp* pointer to the **CANDriver** object

Function Class:

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

10.32.4 Variable Documentation

10.32.4.1 CANDriver CAND1

CAN1 driver identifier.

10.32.5 Define Documentation

10.32.5.1 #define CAN_LIMIT_WARNING 1

Errors rate warning.

10.32.5.2 #define CAN_LIMIT_ERROR 2

Errors rate error.

10.32.5.3 #define CAN_BUS_OFF_ERROR 4

Bus off condition reached.

10.32.5.4 #define CAN_FRAMING_ERROR 8

Framing error of some kind on the CAN bus.

10.32.5.5 #define CAN_OVERFLOW_ERROR 16

Overflow in receive queue.

10.32.5.6 #define CAN_ANY_MAILBOX 0

Special mailbox identifier.

10.32.5.7 #define CAN_USE_SLEEP_MODE TRUE

Sleep mode related APIs inclusion switch.

This option can only be enabled if the CAN implementation supports the sleep mode, see the macro `CAN_SUPPORTS_SLEEP` exported by the underlying implementation.

10.32.5.8 `#define CAN_MAILBOX_TO_MASK(mbx) (1 << ((mbx) - 1))`

Converts a mailbox index to a bit mask.

10.32.5.9 `#define CAN_SUPPORTS_SLEEP TRUE`

This switch defines whether the driver implementation supports a low power switch mode with automatic an wakeup feature.

10.32.5.10 `#define CAN_TX_MAILBOXES 3`

This implementation supports three transmit mailboxes.

10.32.5.11 `#define CAN_RX_MAILBOXES 2`

This implementation supports two receive mailboxes.

10.32.5.12 `#define PLATFORM_CAN_USE_CAN1 FALSE`

CAN1 driver enable switch.

If set to `TRUE` the support for CAN1 is included.

10.32.6 Typedef Documentation

10.32.6.1 `typedef uint32_t canmbx_t`

Type of a transmission mailbox index.

10.32.7 Enumeration Type Documentation

10.32.7.1 `enum canstate_t`

Driver state machine possible states.

Enumerator:

`CAN_UNINIT` Not initialized.

`CAN_STOP` Stopped.

`CAN_STARTING` Starting.

`CAN_READY` Ready.

`CAN_SLEEP` Sleep state.

10.33 EXT Driver

10.33.1 Detailed Description

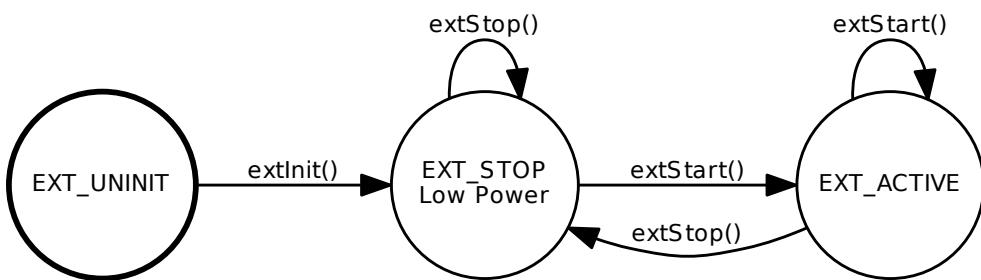
Generic EXT Driver. This module implements a generic EXT (EXternal) driver.

Precondition

In order to use the EXT driver the `HAL_USE_EXT` option must be enabled in `halconf.h`.

10.33.2 Driver State Machine

The driver implements a state machine internally, not all the driver functionalities can be used in any moment, any transition not explicitly shown in the following diagram has to be considered an error and shall be captured by an assertion (if enabled).



10.33.3 EXT Operations.

This driver abstracts generic external interrupt sources, a callback is invoked when a programmable transition is detected on one of the configured channels. Several channel modes are possible.

- `EXT_CH_MODE_DISABLED`, channel not used.
- `EXT_CH_MODE_RISING_EDGE`, callback on a rising edge.
- `EXT_CH_MODE_FALLING_EDGE`, callback on a falling edge.
- `EXT_CH_MODE_BOTH_EDGES`, callback on a both edges.

Data Structures

- struct `EXTChannelConfig`
Channel configuration structure.
- struct `EXTConfig`
Driver configuration structure.
- struct `EXTDriver`
Structure representing an EXT driver.

Functions

- void `extInit` (void)
EXT Driver initialization.
- void `extObjectInit` (`EXTDriver` *extp)
Initializes the standard part of a `EXTDriver` structure.

- void `extStart (EXTDriver *extp, const EXTConfig *config)`
Configures and activates the EXT peripheral.
- void `extStop (EXTDriver *extp)`
Deactivates the EXT peripheral.
- void `extChannelEnable (EXTDriver *extp, expchannel_t channel)`
Enables an EXT channel.
- void `extChannelDisable (EXTDriver *extp, expchannel_t channel)`
Disables an EXT channel.
- void `extSetChannelModel (EXTDriver *extp, expchannel_t channel, const EXTChannelConfig *extcp)`
Changes the operation mode of a channel.
- void `ext_lld_init (void)`
Low level EXT driver initialization.
- void `ext_lld_start (EXTDriver *extp)`
Configures and activates the EXT peripheral.
- void `ext_lld_stop (EXTDriver *extp)`
Deactivates the EXT peripheral.
- void `ext_lld_channel_enable (EXTDriver *extp, expchannel_t channel)`
Enables an EXT channel.
- void `ext_lld_channel_disable (EXTDriver *extp, expchannel_t channel)`
Disables an EXT channel.

Variables

- `EXTDriver EXTD1`
EXT1 driver identifier.

EXT channel modes

- `#define EXT_CH_MODE_EDGES_MASK 3`
Mask of edges field.
- `#define EXT_CH_MODE_DISABLED 0`
Channel disabled.
- `#define EXT_CH_MODE_RISING_EDGE 1`
Rising edge callback.
- `#define EXT_CH_MODE_FALLING_EDGE 2`
Falling edge callback.
- `#define EXT_CH_MODE_BOTH_EDGES 3`
Both edges callback.
- `#define EXT_CH_MODE_AUTOSTART 4`
Channel started automatically on driver start.

Macro Functions

- `#define extChannelEnable(extp, channel) ext_lld_channel_enable(extp, channel)`
Enables an EXT channel.
- `#define extChannelDisable(extp, channel) ext_lld_channel_disable(extp, channel)`
Disables an EXT channel.
- `#define extSetChannelMode(extp, channel, extcp)`
Changes the operation mode of a channel.

Configuration options

- `#define PLATFORM_EXT_USE_EXT1 FALSE`
EXT driver enable switch.

Defines

- `#define EXT_MAX_CHANNELS 20`
Available number of EXT channels.

Typedefs

- `typedef struct EXTDriver EXTDriver`
Type of a structure representing a EXT driver.
- `typedef uint32_t expchannel_t`
EXT channel identifier.
- `typedef void(* extcallback_t)(EXTDriver *extp, expchannel_t channel)`
Type of an EXT generic notification callback.

Enumerations

- `enum extstate_t { EXT_UNINIT = 0, EXT_STOP = 1, EXT_ACTIVE = 2 }`
Driver state machine possible states.

10.33.4 Function Documentation

10.33.4.1 void extInit(void)

EXT Driver initialization.

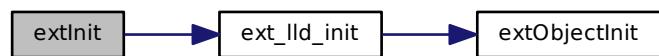
Note

This function is implicitly invoked by `halInit()`, there is no need to explicitly initialize the driver.

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:



10.33.4.2 void extObjectInit (*EXTDriver* * *extp*)

Initializes the standard part of a *EXTDriver* structure.

Parameters

out *extp* pointer to the *EXTDriver* object

Function Class:

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

10.33.4.3 void extStart (*EXTDriver* * *extp*, const *EXTConfig* * *config*)

Configures and activates the EXT peripheral.

Postcondition

After activation all EXT channels are in the disabled state, use *extChannelEnable()* in order to activate them.

Parameters

in	<i>extp</i> pointer to the <i>EXTDriver</i> object
in	<i>config</i> pointer to the <i>EXTConfig</i> 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:

**10.33.4.4 void extStop (*EXTDriver* * *extp*)**

Deactivates the EXT peripheral.

Parameters

in *extp* pointer to the *EXTDriver* 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:



10.33.4.5 void extChannelEnable (EXTDriver * extp, expchannel_t channel)

Enables an EXT channel.

Precondition

The channel must not be in EXT_CH_MODE_DISABLED mode.

Parameters

in	<i>extp</i>	pointer to the <code>EXTDriver</code> object
in	<i>channel</i>	channel to be enabled

Function Class:

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

10.33.4.6 void extChannelDisable (EXTDriver * extp, expchannel_t channel)

Disables an EXT channel.

Precondition

The channel must not be in EXT_CH_MODE_DISABLED mode.

Parameters

in	<i>extp</i>	pointer to the <code>EXTDriver</code> object
in	<i>channel</i>	channel to be disabled

Function Class:

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

10.33.4.7 void extSetChannelModel (EXTDriver * extp, expchannel_t channel, const EXTChannelConfig * extcp)

Changes the operation mode of a channel.

Note

This function attempts to write over the current configuration structure that must have been not declared constant. This violates the `const` qualifier in `extStart()` but it is intentional.

This function cannot be used if the configuration structure is declared `const`.

The effect of this function on constant configuration structures is not defined.

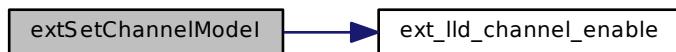
Parameters

in	<i>extp</i>	pointer to the EXTDriver object
in	<i>channel</i>	channel to be changed
in	<i>extcp</i>	new configuration for the channel

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:

**10.33.4.8 void ext_lld_init (void)**

Low level EXT driver initialization.

Function Class:

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

Here is the call graph for this function:

**10.33.4.9 void ext_lld_start (EXTDriver * extp)**

Configures and activates the EXT peripheral.

Parameters

in	<i>extp</i>	pointer to the EXTDriver object
----	-------------	---

Function Class:

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

10.33.4.10 void ext_lld_stop (**EXTDriver** * *extp*)

Deactivates the EXT peripheral.

Parameters

in *extp* pointer to the **EXTDriver** object

Function Class:

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

10.33.4.11 void ext_lld_channel_enable (**EXTDriver** * *extp*, **expchannel_t** *channel*)

Enables an EXT channel.

Parameters

in *extp* pointer to the **EXTDriver** object

in *channel* channel to be enabled

Function Class:

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

10.33.4.12 void ext_lld_channel_disable (**EXTDriver** * *extp*, **expchannel_t** *channel*)

Disables an EXT channel.

Parameters

in *extp* pointer to the **EXTDriver** object

in *channel* channel to be disabled

Function Class:

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

10.33.5 Variable Documentation

10.33.5.1 **EXTDriver EXT1**

EXT1 driver identifier.

10.33.6 Define Documentation

10.33.6.1 #define EXT_CH_MODE_EDGES_MASK 3

Mask of edges field.

10.33.6.2 #define EXT_CH_MODE_DISABLED 0

Channel disabled.

10.33.6.3 #define EXT_CH_MODE_RISING_EDGE 1

Rising edge callback.

10.33.6.4 #define EXT_CH_MODE_FALLING_EDGE 2

Falling edge callback.

10.33.6.5 #define EXT_CH_MODE_BOTH_EDGES 3

Both edges callback.

10.33.6.6 #define EXT_CH_MODE_AUTOSTART 4

Channel started automatically on driver start.

10.33.6.7 #define extChannelEnable(*extp*, *channel*) ext_lld_channel_enable(*extp*, *channel*)

Enables an EXT channel.

Parameters

in	<i>extp</i>	pointer to the EXTDriver object
in	<i>channel</i>	channel to be enabled

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.

10.33.6.8 #define extChannelDisable(*extp*, *channel*) ext_lld_channel_disable(*extp*, *channel*)

Disables an EXT channel.

Parameters

in	<i>extp</i>	pointer to the EXTDriver object
in	<i>channel</i>	channel to be disabled

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.

10.33.6.9 #define extSetChannelMode(*extp*, *channel*, *extcp*)

Value:

```
{
    chSysLock();
    extSetChannelModeI(extp, channel, extcp);
    chSysUnlock();
}
```

Changes the operation mode of a channel.

Note

This function attempts to write over the current configuration structure that must have been not declared constant. This violates the `const` qualifier in `extStart()` but it is intentional. This function cannot be used if the configuration structure is declared `const`.

Parameters

in	<code>extp</code>	pointer to the <code>EXTDriver</code> object
in	<code>channel</code>	channel to be changed
in	<code>extcp</code>	new configuration for the channel

Function Class:

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

10.33.6.10 #define EXT_MAX_CHANNELS 20

Available number of EXT channels.

10.33.6.11 #define PLATFORM_EXT_USE_EXT1 FALSE

EXT driver enable switch.

If set to `TRUE` the support for EXT1 is included.

10.33.7 Typedef Documentation**10.33.7.1 typedef struct EXTDriver EXTDriver**

Type of a structure representing a EXT driver.

10.33.7.2 typedef uint32_t expchannel_t

EXT channel identifier.

10.33.7.3 typedef void(* extcallback_t)(EXTDriver *extp, expchannel_t channel)

Type of an EXT generic notification callback.

Parameters

in	<code>extp</code>	pointer to the <code>EXTDriver</code> object triggering the callback
----	-------------------	--

10.33.8 Enumeration Type Documentation**10.33.8.1 enum extstate_t**

Driver state machine possible states.

Enumerator:

`EXT_UNINIT` Not initialized.

`EXT_STOP` Stopped.

`EXT_ACTIVE` Active.

10.34 GPT Driver

10.34.1 Detailed Description

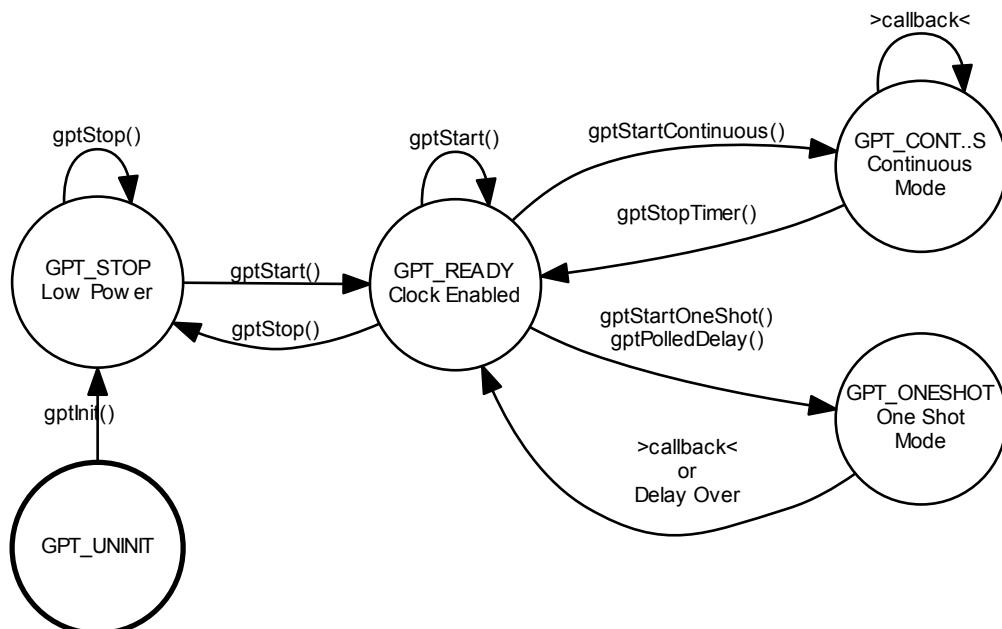
Generic GPT Driver. This module implements a generic GPT (General Purpose Timer) driver. The timer can be programmed in order to trigger callbacks after a specified time period or continuously with a specified interval.

Precondition

In order to use the GPT driver the `HAL_USE_GPT` option must be enabled in `halconf.h`.

10.34.2 Driver State Machine

The driver implements a state machine internally, not all the driver functionalities can be used in any moment, any transition not explicitly shown in the following diagram has to be considered an error and shall be captured by an assertion (if enabled).



10.34.3 GPT Operations.

This driver abstracts a generic timer composed of:

- A clock prescaler.
- A main up counter.
- A comparator register that resets the main counter to zero when the limit is reached. A callback is invoked when this happens.

The timer can operate in three different modes:

- **Continuous Mode**, a periodic callback is invoked until the driver is explicitly stopped.
- **One Shot Mode**, a callback is invoked after the programmed period and then the timer automatically stops.
- **Delay Mode**, the timer is used for inserting a brief delay into the execution flow, no callback is invoked in this mode.

Data Structures

- struct [GPTConfig](#)
Driver configuration structure.
- struct [GPTDriver](#)
Structure representing a GPT driver.

Functions

- void [gptInit](#) (void)
GPT Driver initialization.
- void [gptObjectInit](#) ([GPTDriver](#) *gptp)
Initializes the standard part of a [GPTDriver](#) structure.
- void [gptStart](#) ([GPTDriver](#) *gptp, const [GPTConfig](#) *config)
Configures and activates the GPT peripheral.
- void [gptStop](#) ([GPTDriver](#) *gptp)
Deactivates the GPT peripheral.
- void [gptStartContinuous](#) ([GPTDriver](#) *gptp, [gptcnt_t](#) interval)
Starts the timer in continuous mode.
- void [gptStartContinuousl](#) ([GPTDriver](#) *gptp, [gptcnt_t](#) interval)
Starts the timer in continuous mode.
- void [gptChangeInterval](#) ([GPTDriver](#) *gptp, [gptcnt_t](#) interval)
Changes the interval of GPT peripheral.
- void [gptStartOneShot](#) ([GPTDriver](#) *gptp, [gptcnt_t](#) interval)
Starts the timer in one shot mode.
- void [gptStartOneShotl](#) ([GPTDriver](#) *gptp, [gptcnt_t](#) interval)
Starts the timer in one shot mode.
- void [gptStopTimer](#) ([GPTDriver](#) *gptp)
Stops the timer.
- void [gptStopTimerl](#) ([GPTDriver](#) *gptp)
Stops the timer.
- void [gptPolledDelay](#) ([GPTDriver](#) *gptp, [gptcnt_t](#) interval)
Starts the timer in one shot mode and waits for completion.
- void [gpt_lld_init](#) (void)
Low level GPT driver initialization.
- void [gpt_lld_start](#) ([GPTDriver](#) *gptp)
Configures and activates the GPT peripheral.
- void [gpt_lld_stop](#) ([GPTDriver](#) *gptp)
Deactivates the GPT peripheral.
- void [gpt_lld_start_timer](#) ([GPTDriver](#) *gptp, [gptcnt_t](#) interval)
Starts the timer in continuous mode.
- void [gpt_lld_stop_timer](#) ([GPTDriver](#) *gptp)
Stops the timer.
- void [gpt_lld_polled_delay](#) ([GPTDriver](#) *gptp, [gptcnt_t](#) interval)
Starts the timer in one shot mode and waits for completion.

Variables

- `GPTDriver GPTD1`
GPTD1 driver identifier.

Configuration options

- `#define STM32_GPT_USE_TIM1 FALSE`
GPTD1 driver enable switch.

Defines

- `#define gptChangeInterval(gptp, interval)`
Changes the interval of GPT peripheral.
- `#define gpt_ll_change_interval(gptp, interval)`
Changes the interval of GPT peripheral.

Typedefs

- `typedef struct GPTDriver GPTDriver`
Type of a structure representing a GPT driver.
- `typedef void(* gptcallback_t)(GPTDriver *gptp)`
GPT notification callback type.
- `typedef uint32_t gptfreq_t`
GPT frequency type.
- `typedef uint16_t gptcnt_t`
GPT counter type.

Enumerations

- `enum gptstate_t {`
 `GPT_UNINIT = 0, GPT_STOP = 1, GPT_READY = 2, GPT_CONTINUOUS = 3,`
 `GPT_ONESHOT = 4 }`
Driver state machine possible states.

10.34.4 Function Documentation

10.34.4.1 `void gptInit(void)`

GPT Driver initialization.

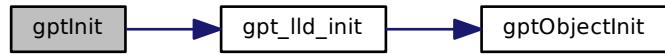
Note

This function is implicitly invoked by `halInit()`, there is no need to explicitly initialize the driver.

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:



10.34.4.2 void gptObjectInit ([GPTDriver](#) * *gptp*)

Initializes the standard part of a [GPTDriver](#) structure.

Parameters

out	<i>gptp</i> pointer to the GPTDriver object
-----	---

Function Class:

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

10.34.4.3 void gptStart ([GPTDriver](#) * *gptp*, const [GPTConfig](#) * *config*)

Configures and activates the GPT peripheral.

Parameters

in	<i>gptp</i> pointer to the GPTDriver object
in	<i>config</i> pointer to the GPTConfig 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:



10.34.4.4 void gptStop ([GPTDriver](#) * *gptp*)

Deactivates the GPT peripheral.

Parameters

in	<i>gptp</i> pointer to the GPTDriver 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:



10.34.4.5 void gptStartContinuous (*GPTDriver* * *gptp*, *gptcnt_t* *interval*)

Starts the timer in continuous mode.

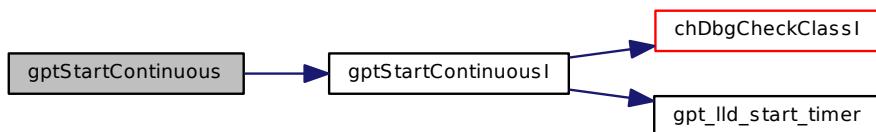
Parameters

in	<i>gptp</i>	pointer to the <i>GPTDriver</i> object
in	<i>interval</i>	period 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:



10.34.4.6 void gptStartContinuousI (*GPTDriver* * *gptp*, *gptcnt_t* *interval*)

Starts the timer in continuous mode.

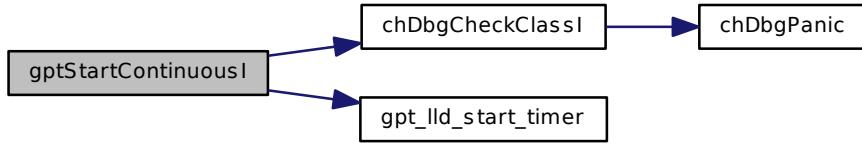
Parameters

in	<i>gptp</i>	pointer to the <i>GPTDriver</i> object
in	<i>interval</i>	period in ticks

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:



10.34.4.7 void gptChangeInterval (`GPTDriver` * *gptp*, `gptcnt_t` *interval*)

Changes the interval of GPT peripheral.

This function changes the interval of a running GPT unit.

Precondition

The GPT unit must have been activated using `gptStart()`.

The GPT unit must have been running in continuous mode using `gptStartContinuous()`.

Postcondition

The GPT unit interval is changed to the new value.

Parameters

in	<i>gptp</i>	pointer to a <code>GPTDriver</code> object
in	<i>interval</i>	new cycle time in timer ticks

Function Class:

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

10.34.4.8 void gptStartOneShot (`GPTDriver` * *gptp*, `gptcnt_t` *interval*)

Starts the timer in one shot mode.

Parameters

in	<i>gptp</i>	pointer to the <code>GPTDriver</code> object
in	<i>interval</i>	time interval 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:



10.34.4.9 void gptStartOneShotl (GPTDriver * *gptp*, gptcnt_t *interval*)

Starts the timer in one shot mode.

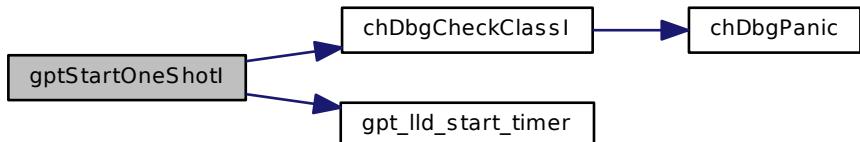
Parameters

in	<i>gptp</i>	pointer to the GPTDriver object
in	<i>interval</i>	time interval 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:



10.34.4.10 void gptStopTimer (GPTDriver * *gptp*)

Stops the timer.

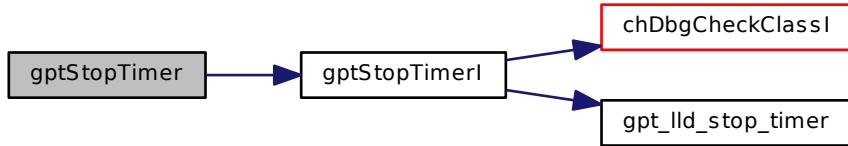
Parameters

in	<i>gptp</i>	pointer to the GPTDriver 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:



10.34.4.11 void gptStopTimerl (GPTDriver * *gptp*)

Stops the timer.

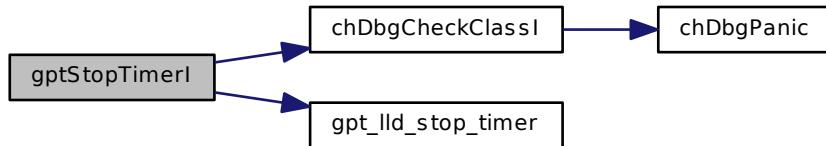
Parameters

in	<i>gptp</i> pointer to the GPTDriver 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:



10.34.4.12 void gptPolledDelay (GPTDriver * *gptp*, gptcnt_t *interval*)

Starts the timer in one shot mode and waits for completion.

This function specifically polls the timer waiting for completion in order to not have extra delays caused by interrupt servicing, this function is only recommended for short delays.

Note

The configured callback is not invoked when using this function.

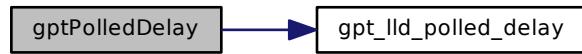
Parameters

in	<i>gptp</i> pointer to the GPTDriver object
in	<i>interval</i> time interval 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:

**10.34.4.13 void gpt_lld_init(void)**

Low level GPT driver initialization.

Function Class:

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

Here is the call graph for this function:

**10.34.4.14 void gpt_lld_start(GPTDriver * gptp)**

Configures and activates the GPT peripheral.

Parameters

in *gptp* pointer to the [GPTDriver](#) object

Function Class:

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

10.34.4.15 void gpt_lld_stop(GPTDriver * gptp)

Deactivates the GPT peripheral.

Parameters

in *gptp* pointer to the [GPTDriver](#) object

Function Class:

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

10.34.4.16 void gpt_lld_start_timer (**GPTDriver** * *gptp*, **gptcnt_t** *interval*)

Starts the timer in continuous mode.

Parameters

in	<i>gptp</i>	pointer to the GPTDriver object
in	<i>interval</i>	period in ticks

Function Class:

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

10.34.4.17 void gpt_lld_stop_timer (**GPTDriver** * *gptp*)

Stops the timer.

Parameters

in	<i>gptp</i>	pointer to the GPTDriver object
----	-------------	--

Function Class:

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

10.34.4.18 void gpt_lld_polled_delay (**GPTDriver** * *gptp*, **gptcnt_t** *interval*)

Starts the timer in one shot mode and waits for completion.

This function specifically polls the timer waiting for completion in order to not have extra delays caused by interrupt servicing, this function is only recommended for short delays.

Parameters

in	<i>gptp</i>	pointer to the GPTDriver object
in	<i>interval</i>	time interval in ticks

Function Class:

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

10.34.5 Variable Documentation

10.34.5.1 **GPTDriver GPTD1**

GPTD1 driver identifier.

10.34.6 Define Documentation

10.34.6.1 #define gptChangelInterval(*gptp*, *interval*)

Value:

```
{
    gpt_lld_change_interval(gptp, interval);
}
\\
```

Changes the interval of GPT peripheral.

This function changes the interval of a running GPT unit.

Precondition

The GPT unit must have been activated using [gptStart\(\)](#).

The GPT unit must have been running in continuous mode using [gptStartContinuous\(\)](#).

Postcondition

The GPT unit interval is changed to the new value.

Parameters

in	<i>gptp</i>	pointer to a GPTDriver object
in	<i>interval</i>	new cycle time in timer ticks

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.

10.34.6.2 #define STM32_GPT_USE_TIM1 FALSE

GPTD1 driver enable switch.

If set to TRUE the support for GPTD1 is included.

Note

The default is TRUE.

10.34.6.3 #define gpt_lld_change_interval(*gptp*, *interval*)

Value:

```
{
    (void)gptp;
    (void)interval;
}
\\
\\
```

Changes the interval of GPT peripheral.

This function changes the interval of a running GPT unit.

Precondition

The GPT unit must have been activated using [gptStart\(\)](#).

The GPT unit must have been running in continuous mode using [gptStartContinuous\(\)](#).

Postcondition

The GPT unit interval is changed to the new value.

Note

The function has effect at the next cycle start.

Parameters

```
in          gptp  pointer to a GPTDriver object
in          interval new cycle time in timer ticks
```

Function Class:

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

10.34.7 Typedef Documentation

10.34.7.1 `typedef struct GPTDriver GPTDriver`

Type of a structure representing a GPT driver.

10.34.7.2 `typedef void(* gptcallback_t)(GPTDriver *gptp)`

GPT notification callback type.

Parameters

```
in          gptp  pointer to a GPTDriver object
```

10.34.7.3 `typedef uint32_t gptfreq_t`

GPT frequency type.

10.34.7.4 `typedef uint16_t gptcnt_t`

GPT counter type.

10.34.8 Enumeration Type Documentation

10.34.8.1 `enum gptstate_t`

Driver state machine possible states.

Enumerator:

GPT_UNINIT Not initialized.

GPT_STOP Stopped.

GPT_READY Ready.

GPT_CONTINUOUS Active in continuous mode.

GPT_ONESHOT Active in one shot mode.

10.35 HAL Driver

10.35.1 Detailed Description

Hardware Abstraction Layer. The HAL (Hardware Abstraction Layer) driver performs the system initialization and includes the platform support code shared by the other drivers. This driver does contain any API function except for a general initialization function `halInit()` that must be invoked before any HAL service can be used, usually the HAL initialization should be performed immediately before the kernel initialization.

Some HAL driver implementations also offer a custom early clock setup function that can be invoked before the C runtime initialization in order to accelerate the startup time.

Functions

- void `halInit` (void)
HAL initialization.
- `bool_t hallsCounterWithin` (`halrtcnt_t` start, `halrtcnt_t` end)
Realtime window test.
- void `halPolledDelay` (`halrtcnt_t` ticks)
Polled delay.
- void `hal_lld_init` (void)
Low level HAL driver initialization.
- void `platform_early_init` (void)
Platform early initialization.

Time conversion utilities for the realtime counter

- #define `S2RTT`(sec) (`halGetCounterFrequency()` * (sec))
Seconds to realtime ticks.
- #define `MS2RTT`(msec) (((`halGetCounterFrequency()` + 999UL) / 1000UL) * (msec))
Milliseconds to realtime ticks.
- #define `US2RTT`(usec)
Microseconds to realtime ticks.
- #define `RTT2S`(ticks) ((ticks) / `halGetCounterFrequency()`)
Realtime ticks to seconds to.
- #define `RTT2MS`(ticks) ((ticks) / (`halGetCounterFrequency()` / 1000UL))
Realtime ticks to milliseconds.
- #define `RTT2US`(ticks) ((ticks) / (`halGetCounterFrequency()` / 1000000UL))
Realtime ticks to microseconds.

Macro Functions

- #define `halGetCounterValue`() `hal_lld_get_counter_value`()
Returns the current value of the system free running counter.
- #define `halGetCounterFrequency`() `hal_lld_get_counter_frequency`()
Realtime counter frequency.

Platform identification

- #define `PLATFORM_NAME` ""

Defines

- #define `HAL_IMPLEMENTES_COUNTERS` TRUE
Defines the support for realtime counters in the HAL.
- #define `hal_lld_get_counter_value`() 0
Returns the current value of the system free running counter.
- #define `hal_lld_get_counter_frequency`() 0
Realtime counter frequency.

Typedefs

- `typedef uint32_t halclock_t`
Type representing a system clock frequency.
- `typedef uint32_t halrtcnt_t`
Type of the realtime free counter value.

10.35.2 Function Documentation

10.35.2.1 void hallinit(void)

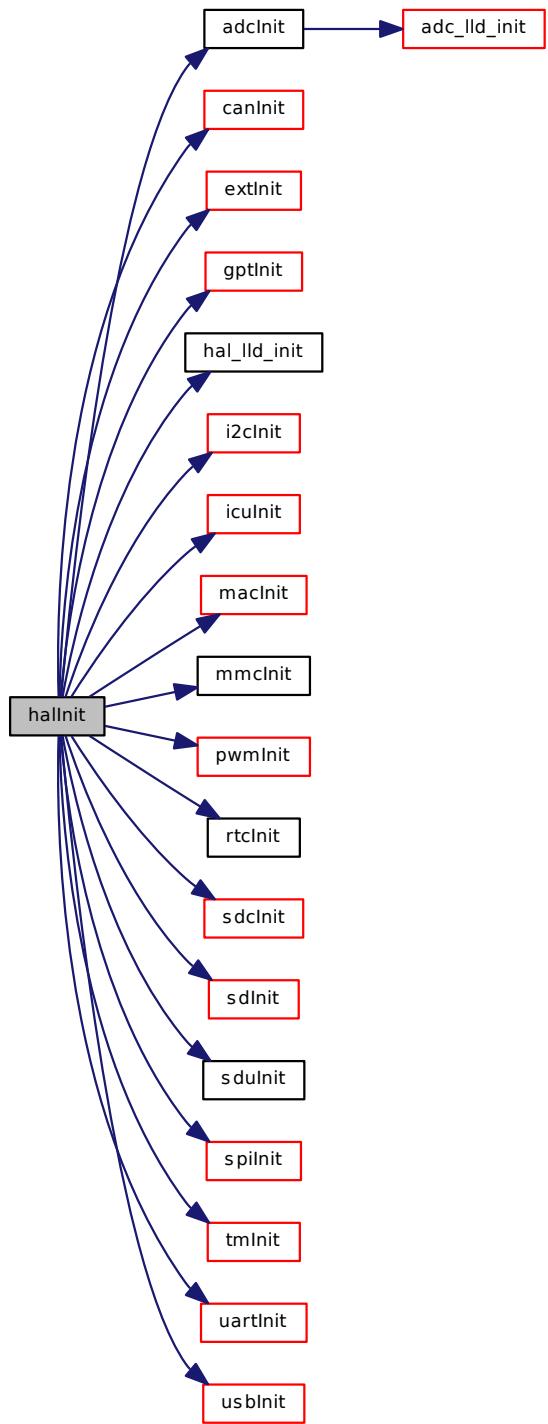
HAL initialization.

This function invokes the low level initialization code then initializes all the drivers enabled in the HAL. Finally the board-specific initialization is performed by invoking `boardInit()` (usually defined in `board.c`).

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:



10.35.2.2 `bool_t hallsCounterWithin(halrtcnt_t start, halrtcnt_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 is an optional service that could not be implemented in all HAL implementations.
This function can be called from any context.

Example 1

Example of a guarded loop using the realtime counter. The loop implements a timeout after one second.

```
halrtcnt_t start = halGetCounterValue();
halrtcnt_t timeout = start + S2RTT(1);
while (my_condition) {
    if (!halIsCounterWithin(start, timeout))
        return TIMEOUT;
    // Do something.
}
// Continue.
```

Example 2

Example of a loop that lasts exactly 50 microseconds.

```
halrtcnt_t start = halGetCounterValue();
halrtcnt_t timeout = start + US2RTT(50);
while (halIsCounterWithin(start, timeout)) {
    // Do something.
}
// Continue.
```

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:

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

10.35.2.3 void halPolledDelay (**halrtcnt_t ticks**)

Polled delay.

Note

The real delays is always few cycles in excess of the specified value.
This is an optional service that could not be implemented in all HAL implementations.
This function can be called from any context.

Parameters

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

Function Class:

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

Here is the call graph for this function:



10.35.2.4 void hal_lld_init(void)

Low level HAL driver initialization.

Function Class:

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

10.35.2.5 void platform_early_init(void)

Platform early initialization.

Note

All the involved constants come from the file `board.h`.

This function is meant to be invoked early during the system initialization, it is usually invoked from the file `board.c`.

Function Class:

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

10.35.3 Define Documentation

10.35.3.1 #define S2RTT(sec)(halGetCounterFrequency() * (sec))

Seconds to realtime ticks.

Converts from seconds to realtime 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.

```
10.35.3.2 #define MS2RTT( msec ) (((halGetCounterFrequency() + 999UL) / 1000UL) * (msec))
```

Milliseconds to realtime ticks.

Converts from milliseconds to realtime 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.

```
10.35.3.3 #define US2RTT( usec )
```

Value:

```
(( (halGetCounterFrequency() + 999999UL) / 1000000UL) * \  
  (usec))
```

Microseconds to realtime ticks.

Converts from microseconds to realtime 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.

```
10.35.3.4 #define RTT2S( ticks ) ((ticks) / halGetCounterFrequency())
```

Realtime ticks to seconds to.

Converts from realtime ticks number to seconds.

Parameters

in *ticks* number of 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.

10.35.3.5 #define RTT2MS(*ticks*) ((ticks) / (halGetCounterFrequency() / 1000UL))

Realtime ticks to milliseconds.

Converts from realtime ticks number to milliseconds.

Parameters

in *ticks* number of 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.

10.35.3.6 #define RTT2US(*ticks*) ((ticks) / (halGetCounterFrequency() / 1000000UL))

Realtime ticks to microseconds.

Converts from realtime ticks number to microseconds.

Parameters

in *ticks* number of 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.

10.35.3.7 #define halGetCounterValue() hal_lld_get_counter_value()

Returns the current value of the system free running counter.

Note

This is an optional service that could not be implemented in all HAL implementations.

This function can be called from any context.

Returns

The value of the system free running counter of type halrtcnt_t.

Function Class:

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

10.35.3.8 #define halGetCounterFrequency() hal_lld_get_counter_frequency()

Realtime counter frequency.

Note

This is an optional service that could not be implemented in all HAL implementations.
This function can be called from any context.

Returns

The realtime counter frequency of type halclock_t.

Function Class:

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

10.35.3.9 #define HAL_IMPLEMENT_COUNTERS TRUE

Defines the support for realtime counters in the HAL.

10.35.3.10 #define hal_ll_get_counter_value() 0

Returns the current value of the system free running counter.

Note

This service is implemented by returning the content of the DWT_CYCCNT register.

Returns

The value of the system free running counter of type halrcnt_t.

Function Class:

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

10.35.3.11 #define hal_ll_get_counter_frequency() 0

Realtime counter frequency.

Note

The DWT_CYCCNT register is incremented directly by the system clock so this function returns STM32_HCLK.

Returns

The realtime counter frequency of type halclock_t.

Function Class:

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

10.35.4 Typedef Documentation

10.35.4.1 typedef uint32_t halclock_t

Type representing a system clock frequency.

10.35.4.2 typedef uint32_t halrcnt_t

Type of the realtime free counter value.

10.36 I2C Driver

10.36.1 Detailed Description

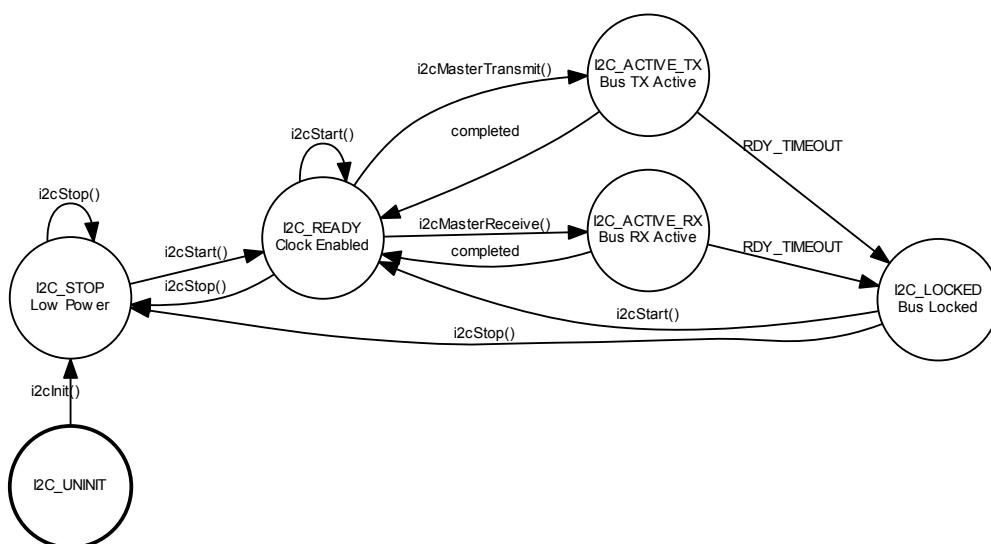
Generic I2C Driver. This module implements a generic I2C (Inter-Integrated Circuit) driver.

Precondition

In order to use the I2C driver the `HAL_USE_I2C` option must be enabled in `halconf.h`.

10.36.2 Driver State Machine

The driver implements a state machine internally, not all the driver functionalities can be used in any moment, any transition not explicitly shown in the following diagram has to be considered an error and shall be captured by an assertion (if enabled).



The driver is not thread safe for performance reasons, if you need to access the I2C bus from multiple threads then use the `i2cAcquireBus()` and `i2cReleaseBus()` APIs in order to gain exclusive access.

Data Structures

- struct `I2CConfig`
Driver configuration structure.
- struct `I2CDriver`
Structure representing an I2C driver.

Functions

- void `i2cInit` (void)
I2C Driver initialization.

- void `i2cObjectInit (I2CDriver *i2cp)`
Initializes the standard part of a `I2CDriver` structure.
- void `i2cStart (I2CDriver *i2cp, const I2CConfig *config)`
Configures and activates the I2C peripheral.
- void `i2cStop (I2CDriver *i2cp)`
Deactivates the I2C peripheral.
- `i2cflags_t i2cGetErrors (I2CDriver *i2cp)`
Returns the errors mask associated to the previous operation.
- `msg_t i2cMasterTransmitTimeout (I2CDriver *i2cp, i2caddr_t addr, const uint8_t *txbuf, size_t txbytes, uint8_t *rdbuf, size_t rxbytes, systime_t timeout)`
Sends data via the I2C bus.
- `msg_t i2cMasterReceiveTimeout (I2CDriver *i2cp, i2caddr_t addr, uint8_t *rdbuf, size_t rxbytes, systime_t timeout)`
Receives data from the I2C bus.
- void `i2cAcquireBus (I2CDriver *i2cp)`
Gains exclusive access to the I2C bus.
- void `i2cReleaseBus (I2CDriver *i2cp)`
Releases exclusive access to the I2C bus.
- void `i2c_lld_init (void)`
Low level I2C driver initialization.
- void `i2c_lld_start (I2CDriver *i2cp)`
Configures and activates the I2C peripheral.
- void `i2c_lld_stop (I2CDriver *i2cp)`
Deactivates the I2C peripheral.
- `msg_t i2c_lld_master_receive_timeout (I2CDriver *i2cp, i2caddr_t addr, uint8_t *rdbuf, size_t rxbytes, systime_t timeout)`
Receives data via the I2C bus as master.
- `msg_t i2c_lld_master_transmit_timeout (I2CDriver *i2cp, i2caddr_t addr, const uint8_t *txbuf, size_t txbytes, uint8_t *rdbuf, size_t rxbytes, systime_t timeout)`
Transmits data via the I2C bus as master.

Variables

- `I2CDriver I2CD1`
I2C1 driver identifier.

I2C bus error conditions

- `#define I2CD_NO_ERROR 0x00`
No error.
- `#define I2CD_BUS_ERROR 0x01`
Bus Error.
- `#define I2CD_ARBITRATION_LOST 0x02`
Arbitration Lost.
- `#define I2CD_ACK_FAILURE 0x04`
Acknowledge Failure.
- `#define I2CD_OVERRUN 0x08`
Overrun/Underrun.
- `#define I2CD_PEC_ERROR 0x10`
PEC Error in reception.
- `#define I2CD_TIMEOUT 0x20`
Hardware timeout.
- `#define I2CD_SMB_ALERT 0x40`
SMBus Alert.

Configuration options

- `#define PLATFORM_I2C_USE_I2C1 FALSE`
I2C1 driver enable switch.

Defines

- `#define I2C_USE_MUTUAL_EXCLUSION TRUE`
Enables the mutual exclusion APIs on the I2C bus.
- `#define i2cMasterTransmit(i2cp, addr, txbuf, txbytes, rxbuf, rxbytes)`
Wrap i2cMasterTransmit function with TIME_INFINITE timeout.
- `#define i2cMasterReceive(i2cp, addr, rxbuf, rxbytes) (i2cMasterReceiveTimeout(i2cp, addr, rxbuf, rxbytes, TIME_INFINITE))`
Wrap i2cMasterReceive function with TIME_INFINITE timeout.
- `#define i2c_lld_get_errors(i2cp) ((i2cp)->errors)`
Get errors from I2C driver.

Typedefs

- `typedef uint16_t i2caddr_t`
Type representing I2C address.
- `typedef uint32_t i2cflags_t`
Type of I2C Driver condition flags.
- `typedef struct I2CDriver I2CDriver`
Type of a structure representing an I2C driver.

Enumerations

- `enum i2cstate_t {`
 `I2C_UNINIT = 0, I2C_STOP = 1, I2C_READY = 2, I2C_ACTIVE_TX = 3,`
 `I2C_ACTIVE_RX = 4 }`
Driver state machine possible states.

10.36.3 Function Documentation

10.36.3.1 void i2cInit(void)

I2C Driver initialization.

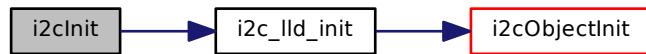
Note

This function is implicitly invoked by `halInit()`, there is no need to explicitly initialize the driver.

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:



10.36.3.2 void i2cObjectInit (I2CDriver * *i2cp*)

Initializes the standard part of a [I2CDriver](#) structure.

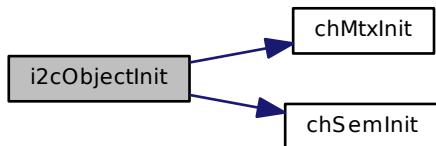
Parameters

out	<i>i2cp</i> pointer to the I2CDriver 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:



10.36.3.3 void i2cStart (I2CDriver * *i2cp*, const I2CConfig * *config*)

Configures and activates the I2C peripheral.

Parameters

in	<i>i2cp</i> pointer to the I2CDriver object
in	<i>config</i> pointer to the I2CConfig 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:



10.36.3.4 void i2cStop (I2CDriver * i2cp)

Deactivates the I2C peripheral.

Parameters

in *i2cp* pointer to the [I2CDriver](#) 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:



10.36.3.5 i2cflags_t i2cGetErrors (I2CDriver * i2cp)

Returns the errors mask associated to the previous operation.

Parameters

in *i2cp* pointer to the [I2CDriver](#) object

Returns

The errors mask.

Function Class:

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

10.36.3.6 msg_t i2cMasterTransmitTimeout (I2CDriver * i2cp, i2caddr_t addr, const uint8_t * txbuf, size_t txbytes, uint8_t * rxbuf, size_t rxbytes, systime_t timeout)

Sends data via the I2C bus.

Function designed to realize "read-through-write" transfer paradigm. If you want transmit data without any further read, than set **rxbytes** field to 0.

Parameters

in	<i>i2cp</i>	pointer to the I2CDriver object
in	<i>addr</i>	slave device address (7 bits) without R/W bit
in	<i>txbuf</i>	pointer to transmit buffer
in	<i>txbytes</i>	number of bytes to be transmitted
out	<i>rxbuf</i>	pointer to receive buffer
in	<i>rxbytes</i>	number of bytes to be received, set it to 0 if you want transmit only
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_INFINITE</i> no timeout.

Returns

The operation status.

Return values

RDY_OK if the function succeeded.

RDY_RESET if one or more I2C errors occurred, the errors can be retrieved using [i2cGetErrors\(\)](#).

RDY_TIMEOUT if a timeout occurred before operation end.

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:



10.36.3.7 **msg_t i2cMasterReceiveTimeout(I2CDriver * i2cp, i2caddr_t addr, uint8_t * rxbuf, size_t rxbytes, systime_t timeout)**

Receives data from the I2C bus.

Parameters

in	<i>i2cp</i>	pointer to the I2CDriver object
in	<i>addr</i>	slave device address (7 bits) without R/W bit
out	<i>rxbuf</i>	pointer to receive buffer
in	<i>rxbytes</i>	number of bytes to be received
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_INFINITE</i> no timeout.

Returns

The operation status.

Return values

RDY_OK if the function succeeded.

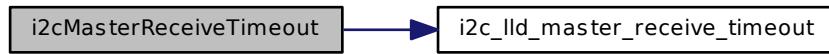
RDY_RESET if one or more I2C errors occurred, the errors can be retrieved using [i2cGetErrors\(\)](#).

RDY_TIMEOUT if a timeout occurred before operation end.

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:

**10.36.3.8 void i2cAcquireBus (I2CDriver * i2cp)**

Gains exclusive access to the I2C bus.

This function tries to gain ownership to the I2C bus, if the bus is already being used then the invoking thread is queued.

Precondition

In order to use this function the option `I2C_USE_MUTUAL_EXCLUSION` must be enabled.

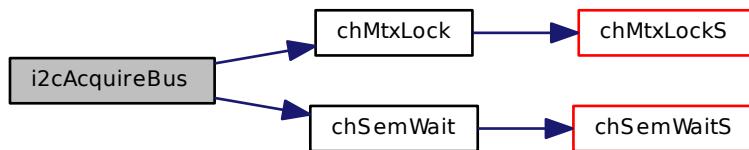
Parameters

in *i2cp* pointer to the `I2CDriver` 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:



10.36.3.9 void i2cReleaseBus (I2CDriver * i2cp)

Releases exclusive access to the I2C bus.

Precondition

In order to use this function the option `I2C_USE_MUTUAL_EXCLUSION` must be enabled.

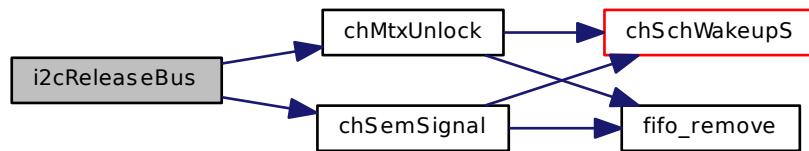
Parameters

in *i2cp* pointer to the `I2CDriver` 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:



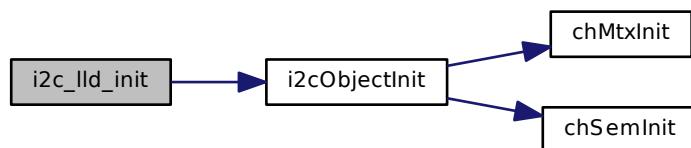
10.36.3.10 void i2c_lld_init (void)

Low level I2C driver initialization.

Function Class:

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

Here is the call graph for this function:



10.36.3.11 void i2c_lld_start (I2CDriver * i2cp)

Configures and activates the I2C peripheral.

Parameters

in *i2cp* pointer to the [I2CDriver](#) object

Function Class:

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

10.36.3.12 void i2c_lld_stop (I2CDriver * *i2cp*)

Deactivates the I2C peripheral.

Parameters

in *i2cp* pointer to the [I2CDriver](#) object

Function Class:

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

10.36.3.13 msg_t i2c_lld_master_receive_timeout (I2CDriver * *i2cp*, i2caddr_t *addr*, uint8_t * *rxbuf*, size_t *rxbytes*, systime_t *timeout*)

Receives data via the I2C bus as master.

Number of receiving bytes must be more than 1 on STM32F1x. This is hardware restriction.

Parameters

in *i2cp* pointer to the [I2CDriver](#) object

in *addr* slave device address

out *rxbuf* pointer to the receive buffer

in *rxbytes* number of bytes to be received

in *timeout* the number of ticks before the operation timeouts, the following special values are allowed:

- *TIME_INFINITE* no timeout.

Returns

The operation status.

Return values

RDY_OK if the function succeeded.

RDY_RESET if one or more I2C errors occurred, the errors can be retrieved using [i2cGetErrors\(\)](#).

RDY_TIMEOUT if a timeout occurred before operation end. **After a timeout the driver must be stopped and restarted because the bus is in an uncertain state.**

Function Class:

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

10.36.3.14 msg_t i2c_lld_master_transmit_timeout (I2CDriver * *i2cp*, i2caddr_t *addr*, const uint8_t * *txbuf*, size_t *txbytes*, uint8_t * *rxbuf*, size_t *rxbytes*, systime_t *timeout*)

Transmits data via the I2C bus as master.

Number of receiving bytes must be 0 or more than 1 on STM32F1x. This is hardware restriction.

Parameters

in	<i>i2cp</i>	pointer to the <code>I2CDriver</code> object
in	<i>addr</i>	slave device address
in	<i>txbuf</i>	pointer to the transmit buffer
in	<i>txbytes</i>	number of bytes to be transmitted
out	<i>rxbuf</i>	pointer to the receive buffer
in	<i>rxbytes</i>	number of bytes to be received
in	<i>timeout</i>	the number of ticks before the operation timeouts, the following special values are allowed:
		• <code>TIME_INFINITE</code> no timeout.

Returns

The operation status.

Return values

`RDY_OK` if the function succeeded.

`RDY_RESET` if one or more I2C errors occurred, the errors can be retrieved using `i2cGetErrors()`.

`RDY_TIMEOUT` if a timeout occurred before operation end. **After a timeout the driver must be stopped and restarted because the bus is in an uncertain state.**

Function Class:

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

10.36.4 Variable Documentation**10.36.4.1 I2CDriver I2CD1**

I2C1 driver identifier.

10.36.5 Define Documentation**10.36.5.1 #define I2CD_NO_ERROR 0x00**

No error.

10.36.5.2 #define I2CD_BUS_ERROR 0x01

Bus Error.

10.36.5.3 #define I2CD_ARBITRATION_LOST 0x02

Arbitration Lost.

10.36.5.4 #define I2CD_ACK_FAILURE 0x04

Acknowledge Failure.

10.36.5.5 #define I2CD_OVERRUN 0x08

Overrun/Underrun.

10.36.5.6 #define I2CD_PEC_ERROR 0x10

PEC Error in reception.

10.36.5.7 #define I2CD_TIMEOUT 0x20

Hardware timeout.

10.36.5.8 #define I2CD_SMB_ALERT 0x40

SMBus Alert.

10.36.5.9 #define I2C_USE_MUTUAL_EXCLUSION TRUE

Enables the mutual exclusion APIs on the I2C bus.

10.36.5.10 #define i2cMasterTransmit(*i2cp*, *addr*, *txbuf*, *txbytes*, *rxbuf*, *rxbytes*)

Value:

```
(i2cMasterTransmitTimeout(i2cp, addr, txbuf, txbytes, rdbuf, rxbytes,      \
TIME_INFINITE))
```

Wrap i2cMasterTransmitTimeout function with TIME_INFINITE timeout.

Function Class:

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

10.36.5.11 #define i2cMasterReceive(*i2cp*, *addr*, *rxbuf*, *rxbytes*) (i2cMasterReceiveTimeout(*i2cp*, *addr*, *rxbuf*, *rxbytes*,
TIME_INFINITE))

Wrap i2cMasterReceiveTimeout function with TIME_INFINITE timeout.

Function Class:

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

10.36.5.12 #define PLATFORM_I2C_USE_I2C1 FALSE

I2C1 driver enable switch.

If set to TRUE the support for I2C1 is included.

Note

The default is FALSE.

10.36.5.13 #define i2c_lld_get_errors(*i2cp*) ((*i2cp*)>errors)

Get errors from I2C driver.

Parameters

in *i2cp* pointer to the [I2CDriver](#) object

Function Class:

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

10.36.6 Typedef Documentation

10.36.6.1 [typedef uint16_t i2caddr_t](#)

Type representing I2C address.

10.36.6.2 [typedef uint32_t i2cflags_t](#)

Type of I2C Driver condition flags.

10.36.6.3 [typedef struct I2CDriver I2CDriver](#)

Type of a structure representing an I2C driver.

10.36.7 Enumeration Type Documentation

10.36.7.1 [enum i2cstate_t](#)

Driver state machine possible states.

Enumerator:

I2C_UNINIT Not initialized.

I2C_STOP Stopped.

I2C_READY Ready.

I2C_ACTIVE_TX Transmitting.

I2C_ACTIVE_RX Receiving.

10.37 I2S Driver

Generic I2S Driver. This module implements a generic I2S driver.

Precondition

In order to use the I2S driver the `HAL_USE_I2S` option must be enabled in [halconf.h](#).

10.37.1 Driver State Machine

10.38 ICU Driver

10.38.1 Detailed Description

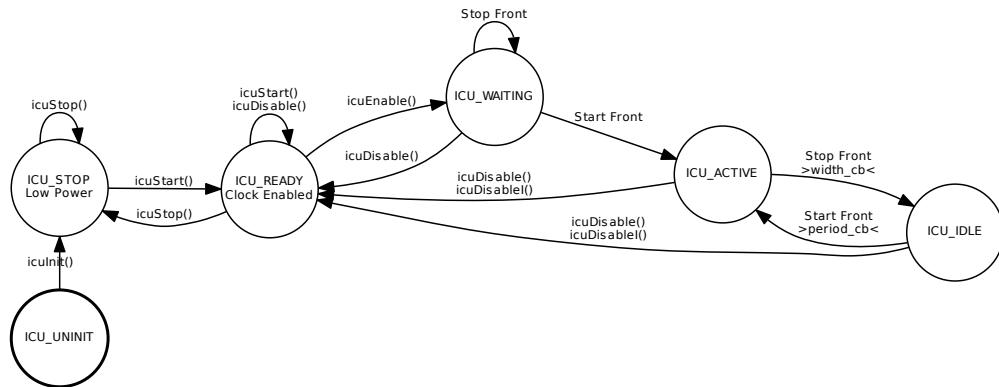
Generic ICU Driver. This module implements a generic ICU (Input Capture Unit) driver.

Precondition

In order to use the ICU driver the `HAL_USE_ICU` option must be enabled in [halconf.h](#).

10.38.2 Driver State Machine

The driver implements a state machine internally, not all the driver functionalities can be used in any moment, any transition not explicitly shown in the following diagram has to be considered an error and shall be captured by an assertion (if enabled).



10.38.3 ICU Operations.

This driver abstracts a generic Input Capture Unit composed of:

- A clock prescaler.
- A main up counter.
- Two capture registers triggered by the rising and falling edges on the sampled input.

The ICU unit can be programmed to synchronize on the rising or falling edge of the sample input:

- **ICU_INPUT_ACTIVE_HIGH**, a rising edge is the start signal.
- **ICU_INPUT_ACTIVE_LOW**, a falling edge is the start signal.

After the activation the ICU unit can be in one of the following states at any time:

- **ICU_WAITING**, waiting the first start signal.
- **ICU_ACTIVE**, after a start signal.
- **ICU_IDLE**, after a stop signal.

Callbacks are invoked when start or stop signals occur.

Data Structures

- struct **ICUConfig**
Driver configuration structure.
- struct **ICUDriver**
Structure representing an ICU driver.

Functions

- void **icuInit** (void)
ICU Driver initialization.
- void **icuObjectInit** (ICUDriver *icup)
Initializes the standard part of a `ICUDriver` structure.
- void **icuStart** (ICUDriver *icup, const ICUConfig *config)
Configures and activates the ICU peripheral.
- void **icuStop** (ICUDriver *icup)
Deactivates the ICU peripheral.
- void **icuEnable** (ICUDriver *icup)
Enables the input capture.
- void **icuDisable** (ICUDriver *icup)
Disables the input capture.
- void **icu_lld_init** (void)
Low level ICU driver initialization.
- void **icu_lld_start** (ICUDriver *icup)
Configures and activates the ICU peripheral.
- void **icu_lld_stop** (ICUDriver *icup)
Deactivates the ICU peripheral.
- void **icu_lld_enable** (ICUDriver *icup)
Enables the input capture.
- void **icu_lld_disable** (ICUDriver *icup)
Disables the input capture.
- **icucnt_t icu_lld_get_width** (ICUDriver *icup)
Returns the width of the latest pulse.
- **icucnt_t icu_lld_get_period** (ICUDriver *icup)
Returns the width of the latest cycle.

Variables

- **ICUDriver ICUD1**
ICU1 driver identifier.

Macro Functions

- #define **icuEnable**(icup) **icu_lld_enable**(icup)
Enables the input capture.
- #define **icuDisable**(icup) **icu_lld_disable**(icup)
Disables the input capture.
- #define **icuGetWidth**(icup) **icu_lld_get_width**(icup)
Returns the width of the latest pulse.
- #define **icuGetPeriod**(icup) **icu_lld_get_period**(icup)
Returns the width of the latest cycle.

Low Level driver helper macros

- #define **_icu_isr_invoke_width_cb**(icup)
Common ISR code, ICU width event.
- #define **_icu_isr_invoke_period_cb**(icup)
Common ISR code, ICU period event.
- #define **_icu_isr_invoke_overflow_cb**(icup)
Common ISR code, ICU timer overflow event.

Configuration options

- `#define PLATFORM_ICU_USE_ICU1 FALSE`
ICU driver enable switch.

Typedefs

- `typedef struct ICUDriver ICUDriver`
Type of a structure representing an ICU driver.
- `typedef void(* icucallback_t)(ICUDriver *icup)`
ICU notification callback type.
- `typedef uint32_t icufreq_t`
ICU frequency type.
- `typedef uint16_t icucnt_t`
ICU counter type.

Enumerations

- `enum icustate_t {`
`ICU_UNINIT = 0, ICU_STOP = 1, ICU_READY = 2, ICU_WAITING = 3,`
`ICU_ACTIVE = 4, ICU_IDLE = 5 }`
Driver state machine possible states.
- `enum icumode_t { ICU_INPUT_ACTIVE_HIGH = 0, ICU_INPUT_ACTIVE_LOW = 1 }`
ICU driver mode.

10.38.4 Function Documentation

10.38.4.1 void icuInit(void)

ICU Driver initialization.

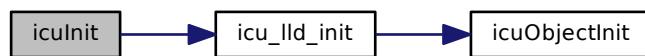
Note

This function is implicitly invoked by `halInit()`, there is no need to explicitly initialize the driver.

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:



10.38.4.2 void icuObjectInit (ICUDriver * *icup*)

Initializes the standard part of a [ICUDriver](#) structure.

Parameters

out *icup* pointer to the [ICUDriver](#) object

Function Class:

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

10.38.4.3 void icuStart (ICUDriver * *icup*, const ICUConfig * *config*)

Configures and activates the ICU peripheral.

Parameters

in	<i>icup</i> pointer to the ICUDriver object
in	<i>config</i> pointer to the ICUConfig 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:

**10.38.4.4 void icuStop (ICUDriver * *icup*)**

Deactivates the ICU peripheral.

Parameters

in *icup* pointer to the [ICUDriver](#) 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:



10.38.4.5 void icuEnable (ICUDriver * *icup*)

Enables the input capture.

Parameters

in *icup* pointer to the [ICUDriver](#) 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:



10.38.4.6 void icuDisable (ICUDriver * *icup*)

Disables the input capture.

Parameters

in *icup* pointer to the [ICUDriver](#) 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:



10.38.4.7 void icu_lld_init(void)

Low level ICU driver initialization.

Function Class:

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

Here is the call graph for this function:



10.38.4.8 void icu_lld_start(ICUDriver * icup)

Configures and activates the ICU peripheral.

Parameters

in *icup* pointer to the [ICUDriver](#) object

Function Class:

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

10.38.4.9 void icu_lld_stop(ICUDriver * icup)

Deactivates the ICU peripheral.

Parameters

in *icup* pointer to the [ICUDriver](#) object

Function Class:

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

10.38.4.10 void icu_lld_enable (**ICUDriver** * *icup*)

Enables the input capture.

Parameters

in *icup* pointer to the **ICUDriver** object

Function Class:

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

10.38.4.11 void icu_lld_disable (**ICUDriver** * *icup*)

Disables the input capture.

Parameters

in *icup* pointer to the **ICUDriver** object

Function Class:

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

10.38.4.12 icucnt_t icu_lld_get_width (**ICUDriver** * *icup*)

Returns the width of the latest pulse.

The pulse width is defined as number of ticks between the start edge and the stop edge.

Parameters

in *icup* pointer to the **ICUDriver** object

Returns

The number of ticks.

Function Class:

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

10.38.4.13 icucnt_t icu_lld_get_period (**ICUDriver** * *icup*)

Returns the width of the latest cycle.

The cycle width is defined as number of ticks between a start edge and the next start edge.

Parameters

in *icup* pointer to the **ICUDriver** object

Returns

The number of ticks.

Function Class:

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

10.38.5 Variable Documentation

10.38.5.1 ICUDriver ICUD1

ICU1 driver identifier.

10.38.6 Define Documentation

10.38.6.1 #define icuEnable(*icup*) icu_lld_enable(*icup*)

Enables the input capture.

Parameters

in *icup* pointer to the [ICUDriver](#) object

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.

10.38.6.2 #define icuDisable(*icup*) icu_lld_disable(*icup*)

Disables the input capture.

Parameters

in *icup* pointer to the [ICUDriver](#) object

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.

10.38.6.3 #define icuGetWidth(*icup*) icu_lld_get_width(*icup*)

Returns the width of the latest pulse.

The pulse width is defined as number of ticks between the start edge and the stop edge.

Note

This function is meant to be invoked from the width capture callback only.

Parameters

in *icup* pointer to the [ICUDriver](#) object

Returns

The number of ticks.

Function Class:

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

10.38.6.4 #define icuGetPeriod(*icup*) icu_lld_get_period(*icup*)

Returns the width of the latest cycle.

The cycle width is defined as number of ticks between a start edge and the next start edge.

Note

This function is meant to be invoked from the width capture callback only.

Parameters

in *icup* pointer to the [ICUDriver](#) object

Returns

The number of ticks.

Function Class:

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

10.38.6.5 #define _icu_isr_invoke_width_cb(*icup*)

Value:

```
{
    if ((icup)->state != ICU_WAITING) \
        (icup)->state = ICU_IDLE;
        (icup)->config->width_cb(icup);
    }
}
```

Common ISR code, ICU width event.

Parameters

in *icup* pointer to the [ICUDriver](#) object

Function Class:

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

10.38.6.6 #define _icu_isr_invoke_period_cb(*icup*)

Value:

```
{
    icustate_t previous_state = (icup)->state;
    (icup)->state = ICU_ACTIVE;
    if (previous_state != ICU_WAITING)
        (icup)->config->period_cb(icup);
}
```

Common ISR code, ICU period event.

Parameters

in *icup* pointer to the [ICUDriver](#) object

Function Class:

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

```
10.38.6.7 #define _icu_isr_invoke_overflow_cb( icup )
```

Value:

```
{\n    (icup)->config->overflow_cb(icup);\n}
```

Common ISR code, ICU timer overflow event.

Parameters

in *icup* pointer to the [ICUDriver](#) object

Function Class:

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

```
10.38.6.8 #define PLATFORM_ICU_USE_ICU1 FALSE
```

ICU driver enable switch.

If set to TRUE the support for ICU1 is included.

10.38.7 Typedef Documentation

```
10.38.7.1 typedef struct ICUDriver ICUDriver
```

Type of a structure representing an ICU driver.

```
10.38.7.2 typedef void(* icucallback_t)(ICUDriver *icup)
```

ICU notification callback type.

Parameters

in *icup* pointer to a [ICUDriver](#) object

```
10.38.7.3 typedef uint32_t icufreq_t
```

ICU frequency type.

```
10.38.7.4 typedef uint16_t icucnt_t
```

ICU counter type.

10.38.8 Enumeration Type Documentation

```
10.38.8.1 enum icustate_t
```

Driver state machine possible states.

Enumerator:

ICU_UNINIT Not initialized.

ICU_STOP Stopped.

ICU_READY Ready.

ICU_WAITING Waiting first edge.

ICU_ACTIVE Active cycle phase.

ICU_IDLE Idle cycle phase.

10.38.8.2 enum icumode_t

ICU driver mode.

Enumerator:

ICU_INPUT_ACTIVE_HIGH Trigger on rising edge.

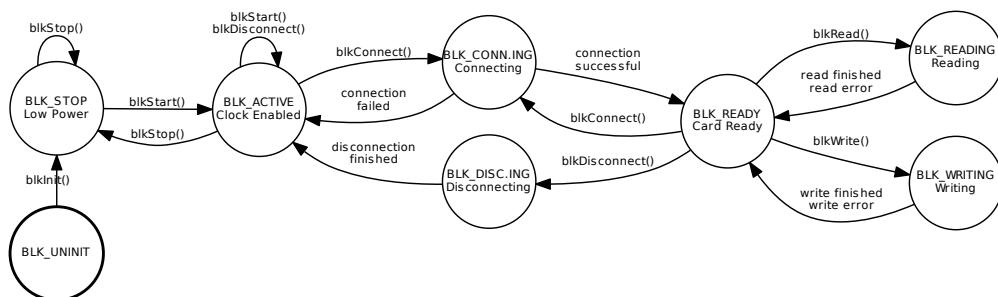
ICU_INPUT_ACTIVE_LOW Trigger on falling edge.

10.39 Abstract I/O Block Device

10.39.1 Detailed Description

10.39.2 Driver State Machine

The drivers implementing this interface shall implement the following state machine internally. Not all the driver functionalities can be used in any moment, any transition not explicitly shown in the following diagram has to be considered an error and shall be captured by an assertion (if enabled).



This module defines an abstract interface for accessing generic block devices.

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 block devices independent from the implementation logic.

Data Structures

- struct [BlockDeviceInfo](#)
Block device info.
- struct [BaseBlockDeviceVMT](#)

- struct `BaseBlockDevice`
Base block device class.

Macro Functions (`BaseBlockDevice`)

- `#define blkGetDriverState(ip) ((ip)->state)`
Returns the driver state.
- `#define blkIsTransferring(ip)`
Determines if the device is transferring data.
- `#define blkIsInserted(ip) ((ip)->vmt->is_inserted(ip))`
Returns the media insertion status.
- `#define blkIsWriteProtected(ip) ((ip)->vmt->is_protected(ip))`
Returns the media write protection status.
- `#define blkConnect(ip) ((ip)->vmt->connect(ip))`
Performs the initialization procedure on the block device.
- `#define blkDisconnect(ip) ((ip)->vmt->disconnect(ip))`
Terminates operations on the block device.
- `#define blkRead(ip, startblk, buf, n) ((ip)->vmt->read(ip, startblk, buf, n))`
Reads one or more blocks.
- `#define blkWrite(ip, startblk, buf, n) ((ip)->vmt->write(ip, startblk, buf, n))`
Writes one or more blocks.
- `#define blkSync(ip) ((ip)->vmt->sync(ip))`
Ensures write synchronization.
- `#define blkGetInfo(ip, bdip) ((ip)->vmt->get_info(ip, bdip))`
Returns a media information structure.

Defines

- `#define _base_block_device_methods`
`BaseBlockDevice` specific methods.
- `#define _base_block_device_data`
`BaseBlockDevice` specific data.

Enumerations

- enum `blkstate_t` {

`BLK_UNINIT` = 0, `BLK_STOP` = 1, `BLK_ACTIVE` = 2, `BLK_CONNECTING` = 3,
`BLK_DISCONNECTING` = 4, `BLK_READY` = 5, `BLK_READING` = 6, `BLK_WRITING` = 7,
`BLK_SYNCING` = 8 }

Driver state machine possible states.

10.39.3 Define Documentation

10.39.3.1 `#define _base_block_device_methods`

Value:

```
/* Removable media detection.*/
bool_t (*is_inserted)(void *instance);
/* Removable write protection detection.*/
bool_t (*is_protected)(void *instance);
/* Connection to the block device.*/
bool_t (*connect)(void *instance);
/* Disconnection from the block device.*/
bool_t (*disconnect)(void *instance);
/* Reads one or more blocks.*/
bool_t (*read)(void *instance, uint32_t startblk,
               uint8_t *buffer, uint32_t n);
/* Writes one or more blocks.*/
bool_t (*write)(void *instance, uint32_t startblk,
                const uint8_t *buffer, uint32_t n);
/* Write operations synchronization.*/
bool_t (*sync)(void *instance);
/* Obtains info about the media.*/
bool_t (*get_info)(void *instance, BlockDeviceInfo *bdip);
```

[BaseBlockDevice](#) specific methods.

10.39.3.2 #define _base_block_device_data

Value:

```
/* Driver state.*/
blkstate_t state;
```

[BaseBlockDevice](#) specific data.

10.39.3.3 #define blkGetDriverState(ip) ((ip)->state)

Returns the driver state.

Note

Can be called in ISR context.

Parameters

in	<i>ip</i> pointer to a BaseBlockDevice or derived class
----	---

Returns

The driver state.

Function Class:

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

10.39.3.4 #define blkIsTransferring(ip)

Value:

```
((((ip)->state) == BLK_CONNECTING) ||
   (((ip)->state) == BLK_DISCONNECTING) ||
   (((ip)->state) == BLK_READING) ||
   (((ip)->state) == BLK_WRITING))
```

Determines if the device is transferring data.

Note

Can be called in ISR context.

Parameters

in *ip* pointer to a `BaseBlockDevice` or derived class

Returns

The driver state.

Return values

FALSE the device is not transferring data.

TRUE the device not transferring data.

Function Class:

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

10.39.3.5 #define blkIsInserted(*ip*) ((*ip*)->vmt->is_inserted(*ip*))

Returns the media insertion status.

Note

On some implementations this function can only be called if the device is not transferring data. The function `blkIsTransferring()` should be used before calling this function.

Parameters

in *ip* pointer to a `BaseBlockDevice` or derived class

Returns

The media state.

Return values

FALSE media not inserted.

TRUE media inserted.

Function Class:

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

10.39.3.6 #define blkIsWriteProtected(*ip*) ((*ip*)->vmt->is_protected(*ip*))

Returns the media write protection status.

Parameters

in *ip* pointer to a `BaseBlockDevice` or derived class

Returns

The media state.

Return values

FALSE writable media.

TRUE non writable media.

Function Class:

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

10.39.3.7 #define blkConnect(*ip*) ((*ip*)->vmt->connect(*ip*))

Performs the initialization procedure on the block device.

This function should be performed before I/O operations can be attempted on the block device and after insertion has been confirmed using `blkIsInserted()`.

Parameters

in	<i>ip</i>	pointer to a <code>BaseBlockDevice</code> or derived class
----	-----------	--

Returns

The operation status.

Return values

<code>CH_SUCCESS</code>	operation succeeded.
-------------------------	----------------------

<code>CH_FAILED</code>	operation failed.
------------------------	-------------------

Function Class:

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

10.39.3.8 #define blkDisconnect(*ip*) ((*ip*)->vmt->disconnect(*ip*))

Terminates operations on the block device.

This operation safely terminates operations on the block device.

Parameters

in	<i>ip</i>	pointer to a <code>BaseBlockDevice</code> or derived class
----	-----------	--

Returns

The operation status.

Return values

<code>CH_SUCCESS</code>	operation succeeded.
-------------------------	----------------------

<code>CH_FAILED</code>	operation failed.
------------------------	-------------------

Function Class:

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

10.39.3.9 #define blkRead(*ip*, *startblk*, *buf*, *n*) ((*ip*)->vmt->read(*ip*, *startblk*, *buf*, *n*))

Reads one or more blocks.

Parameters

in	<i>ip</i>	pointer to a <code>BaseBlockDevice</code> or derived class
in	<i>startblk</i>	first block to read
out	<i>buf</i>	pointer to the read buffer
in	<i>n</i>	number of blocks to read

Returns

The operation status.

Return values

<i>CH_SUCCESS</i>	operation succeeded.
<i>CH_FAILED</i>	operation failed.

Function Class:

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

10.39.3.10 #define blkWrite(*ip*, *startblk*, *buf*, *n*) ((*ip*)->vmt->write(*ip*, *startblk*, *buf*, *n*))

Writes one or more blocks.

Parameters

<i>in</i>	<i>ip</i> pointer to a BaseBlockDevice or derived class
<i>in</i>	<i>startblk</i> first block to write
<i>out</i>	<i>buf</i> pointer to the write buffer
<i>in</i>	<i>n</i> number of blocks to write

Returns

The operation status.

Return values

<i>CH_SUCCESS</i>	operation succeeded.
<i>CH_FAILED</i>	operation failed.

Function Class:

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

10.39.3.11 #define blkSync(*ip*) ((*ip*)->vmt->sync(*ip*))

Ensures write synchronization.

Parameters

<i>in</i>	<i>ip</i> pointer to a BaseBlockDevice or derived class
-----------	---

Returns

The operation status.

Return values

<i>CH_SUCCESS</i>	operation succeeded.
<i>CH_FAILED</i>	operation failed.

Function Class:

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

10.39.3.12 #define blkGetInfo(*ip*, *bdip*) ((*ip*)->vmt->get_info(*ip*, *bdip*))

Returns a media information structure.

Parameters

<i>in</i>	<i>ip</i> pointer to a BaseBlockDevice or derived class
<i>out</i>	<i>bdip</i> pointer to a BlockDeviceInfo structure

Returns

The operation status.

Return values

CH_SUCCESS operation succeeded.
CH_FAILED operation failed.

Function Class:

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

10.39.4 Enumeration Type Documentation

10.39.4.1 enum blkstate_t

Driver state machine possible states.

Enumerator:

BLK_UNINIT Not initialized.
BLK_STOP Stopped.
BLK_ACTIVE Interface active.
BLK_CONNECTING Connection in progress.
BLK_DISCONNECTING Disconnection in progress.
BLK_READY Device ready.
BLK_READING Read operation in progress.
BLK_WRITING Write operation in progress.
BLK_SYNCING Sync. operation in progress.

10.40 Abstract I/O Channel

10.40.1 Detailed Description

This module defines an abstract interface for I/O channels by extending the [BaseSequentialStream](#) interface.

Note that no code is present, I/O channels are just abstract interface like structures, you should look at the systems as to a set of abstract C++ classes (even if written in C). Specific device drivers can use/extend the interface and implement them.

This system has the advantage to make the access to channels independent from the implementation logic.

Data Structures

- struct [BaseChannelVMT](#)
 BaseChannel virtual methods table.
- struct [BaseChannel](#)
 Base channel class.
- struct [BaseAsynchronousChannelVMT](#)
 BaseAsynchronousChannel virtual methods table.
- struct [BaseAsynchronousChannel](#)
 Base asynchronous channel class.

Macro Functions (BaseChannel)

- #define `chnPutTimeout(ip, b, time)` ((ip)->vmt->putt(ip, b, time))

Channel blocking byte write with timeout.
- #define `chnGetTimeout(ip, time)` ((ip)->vmt->gett(ip, time))

Channel blocking byte read with timeout.
- #define `chnWrite(ip, bp, n)` chSequentialStreamWrite(ip, bp, n)

Channel blocking write.
- #define `chnWriteTimeout(ip, bp, n, time)` ((ip)->vmt->writet(ip, bp, n, time))

Channel blocking write with timeout.
- #define `chnRead(ip, bp, n)` chSequentialStreamRead(ip, bp, n)

Channel blocking read.
- #define `chnReadTimeout(ip, bp, n, time)` ((ip)->vmt->readt(ip, bp, n, time))

Channel blocking read with timeout.

I/O status flags added to the event listener

- #define `CHN_NO_ERROR` 0

No pending conditions.
- #define `CHN_CONNECTED` 1

Connection happened.
- #define `CHN_DISCONNECTED` 2

Disconnection happened.
- #define `CHN_INPUT_AVAILABLE` 4

Data available in the input queue.
- #define `CHN_OUTPUT_EMPTY` 8

Output queue empty.
- #define `CHN_TRANSMISSION_END` 16

Transmission end.

Macro Functions (BaseAsynchronousChannel)

- #define `chnGetEventSource(ip)` (&((ip)->event))

Returns the I/O condition event source.
- #define `chnAddFlagsI(ip, flags)`

Adds status flags to the listeners's flags mask.

Defines

- #define `_base_channel_methods`

BaseChannel specific methods.
- #define `_base_channel_data _base_sequential_stream_data`

BaseChannel specific data.
- #define `_base_asynchronous_channel_methods _base_channel_methods \`

BaseAsynchronousChannel specific methods.
- #define `_base_asynchronous_channel_data`

BaseAsynchronousChannel specific data.

10.40.2 Define Documentation

10.40.2.1 #define _base_channel_methods

Value:

```
_base_sequential_stream_methods \
/* Channel put method with timeout specification.*/
msg_t (*putt)(void *instance, uint8_t b, systime_t time); \
/* Channel get method with timeout specification.*/
msg_t (*gett)(void *instance, systime_t time); \
/* Channel write method with timeout specification.*/
size_t (*writet)(void *instance, const uint8_t *bp, \
                  size_t n, systime_t time); \
/* Channel read method with timeout specification.*/
size_t (*readt)(void *instance, uint8_t *bp, size_t n, systime_t time);
```

[BaseChannel](#) specific methods.

10.40.2.2 #define _base_channel_data _base_sequential_stream_data

[BaseChannel](#) specific data.

Note

It is empty because [BaseChannel](#) is only an interface without implementation.

10.40.2.3 #define chnPutTimeout(ip, b, time) ((ip)->vmt->putt(ip, b, time))

Channel blocking byte write with timeout.

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 BaseChannel or derived class
in	<i>b</i>	the byte value to be written to the channel
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

The operation status.

Return values

<i>Q_OK</i>	if the operation succeeded.
<i>Q_TIMEOUT</i>	if the specified time expired.
<i>Q_RESET</i>	if the channel associated queue (if any) was reset.

Function Class:

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

10.40.2.4 #define chnGetTimeout(*ip*, *time*) ((*ip*)->vmt->gett(*ip*, *time*))

Channel blocking byte read with timeout.

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 BaseChannel or derived class
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 byte value from the queue.

Return values

<i>Q_TIMEOUT</i>	if the specified time expired.
<i>Q_RESET</i>	if the channel associated queue (if any) has been reset.

Function Class:

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

10.40.2.5 #define chnWrite(*ip*, *bp*, *n*) chSequentialStreamWrite(*ip*, *bp*, *n*)

Channel blocking write.

The function writes data from a buffer 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 BaseChannel 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.

Function Class:

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

10.40.2.6 #define chnWriteTimeout(*ip*, *bp*, *n*, *time*) ((*ip*)->vmt->writet(*ip*, *bp*, *n*, *time*))

Channel blocking write with timeout.

The function writes data from a buffer 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 BaseChannel or derived class
out	<i>bp</i>	pointer to the data buffer
in	<i>n</i>	the maximum amount of data to be transferred

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

The number of bytes transferred.

Function Class:

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

10.40.2.7 #define chnRead(ip, bp, n) chSequentialStreamRead(ip, bp, n)

Channel blocking read.

The function reads data from a channel into a buffer. If the data is not available then the calling thread is suspended.

Parameters

in	<i>ip</i> pointer to a BaseChannel 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.

Function Class:

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

10.40.2.8 #define chnReadTimeout(ip, bp, n, time) ((ip)->vmt->readt(ip, bp, n, time))

Channel blocking read with timeout.

The function reads data from a channel into a buffer. If the data is not available then the calling thread is suspended.

Parameters

in	<i>ip</i> pointer to a BaseChannel or derived class
in	<i>bp</i> pointer to the data buffer
in	<i>n</i> the maximum amount of data to be transferred
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

The number of bytes transferred.

Function Class:

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

10.40.2.9 #define CHN_NO_ERROR 0

No pending conditions.

10.40.2.10 #define CHN_CONNECTED 1

Connection happened.

10.40.2.11 #define CHN_DISCONNECTED 2

Disconnection happened.

10.40.2.12 #define CHN_INPUT_AVAILABLE 4

Data available in the input queue.

10.40.2.13 #define CHN_OUTPUT_EMPTY 8

Output queue empty.

10.40.2.14 #define CHN_TRANSMISSION_END 16

Transmission end.

10.40.2.15 #define _base_asynchronous_channel_methods _base_channel_methods \

[BaseAsynchronousChannel](#) specific methods.

10.40.2.16 #define _base_asynchronous_channel_data

Value:

```
_base_channel_data
/* I/O condition event source.*/
EventSource           event;
```

[BaseAsynchronousChannel](#) specific data.

10.40.2.17 #define chnGetEventSource(*ip*) (&(*ip*)>event)

Returns the I/O condition event source.

The event source is broadcasted when an I/O condition happens.

Parameters

in *ip* pointer to a [BaseAsynchronousChannel](#) or derived class

Returns

A pointer to an [EventSource](#) object.

Function Class:

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

10.40.2.18 #define chnAddFlags(ip, flags)

Value:

```
{
    chEvtBroadcastFlagsI(&(ip)->event, flags);
} \\
```

Adds status flags to the listeners's flags mask.

This function is usually called from the I/O ISRs in order to notify I/O conditions such as data events, errors, signal changes etc.

Parameters

in	<i>ip</i>	pointer to a BaseAsynchronousChannel or derived class
in	<i>flags</i>	condition flags 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.

10.41 MAC Driver

10.41.1 Detailed Description

Generic MAC driver. This module implements a generic MAC (Media Access Control) driver for Ethernet controllers.

Precondition

In order to use the MAC driver the `HAL_USE_MAC` option must be enabled in [halconf.h](#).

Data Structures

- struct [MACConfig](#)
Driver configuration structure.
- struct [MACDriver](#)
Structure representing a MAC driver.
- struct [MACTransmitDescriptor](#)
Structure representing a transmit descriptor.
- struct [MACReceiveDescriptor](#)
Structure representing a receive descriptor.

Functions

- void [macInit](#) (void)
MAC Driver initialization.
- void [macObjectInit](#) ([MACDriver](#) *macp)
Initialize the standard part of a [MACDriver](#) structure.
- void [macStart](#) ([MACDriver](#) *macp, const [MACConfig](#) *config)
Configures and activates the MAC peripheral.
- void [macStop](#) ([MACDriver](#) *macp)
Deactivates the MAC peripheral.
- [msg_t](#) [macWaitTransmitDescriptor](#) ([MACDriver](#) *macp, [MACTransmitDescriptor](#) *tdp, [systime_t](#) time)

- **Allocates a transmission descriptor.**
- void **macReleaseTransmitDescriptor** (MACTransmitDescriptor *tdp)

Releases a transmit descriptor and starts the transmission of the enqueued data as a single frame.
- msg_t **macWaitReceiveDescriptor** (MACDriver *macp, MACReceiveDescriptor *rdp, systime_t time)

Waits for a received frame.
- void **macReleaseReceiveDescriptor** (MACReceiveDescriptor *rdp)

Releases a receive descriptor.
- bool_t **macPollLinkStatus** (MACDriver *macp)

Updates and returns the link status.
- void **mac_lld_init** (void)

Low level MAC initialization.
- void **mac_lld_start** (MACDriver *macp)

Configures and activates the MAC peripheral.
- void **mac_lld_stop** (MACDriver *macp)

Deactivates the MAC peripheral.
- msg_t **mac_lld_get_transmit_descriptor** (MACDriver *macp, MACTransmitDescriptor *tdp)

Returns a transmission descriptor.
- void **mac_lld_release_transmit_descriptor** (MACTransmitDescriptor *tdp)

Releases a transmit descriptor and starts the transmission of the enqueued data as a single frame.
- msg_t **mac_lld_get_receive_descriptor** (MACDriver *macp, MACReceiveDescriptor *rdp)

Returns a receive descriptor.
- void **mac_lld_release_receive_descriptor** (MACReceiveDescriptor *rdp)

Releases a receive descriptor.
- bool_t **mac_lld_poll_link_status** (MACDriver *macp)

Updates and returns the link status.
- size_t **mac_lld_write_transmit_descriptor** (MACTransmitDescriptor *tdp, uint8_t *buf, size_t size)

Writes to a transmit descriptor's stream.
- size_t **mac_lld_read_receive_descriptor** (MACReceiveDescriptor *rdp, uint8_t *buf, size_t size)

Reads from a receive descriptor's stream.
- uint8_t * **mac_lld_get_next_transmit_buffer** (MACTransmitDescriptor *tdp, size_t size, size_t *sizep)

Returns a pointer to the next transmit buffer in the descriptor chain.
- const uint8_t * **mac_lld_get_next_receive_buffer** (MACReceiveDescriptor *rdp, size_t *sizep)

Returns a pointer to the next receive buffer in the descriptor chain.

Variables

- **MACDriver ETHD1**

MAC1 driver identifier.

MAC configuration options

- #define **MAC_USE_ZERO_COPY** FALSE

Enables an event sources for incoming packets.
- #define **MAC_USE_EVENTS** TRUE

Enables an event sources for incoming packets.

Macro Functions

- `#define macGetReceiveEventSource(macp) (&(macp)->rdevent)`
Returns the received frames event source.
- `#define macWriteTransmitDescriptor(tdp, buf, size) mac_lld_write_transmit_descriptor(tdp, buf, size)`
Writes to a transmit descriptor's stream.
- `#define macReadReceiveDescriptor(rdp, buf, size) mac_lld_read_receive_descriptor(rdp, buf, size)`
Reads from a receive descriptor's stream.
- `#define macGetNextTransmitBuffer(tdp, size, sizep) mac_lld_get_next_transmit_buffer(tdp, size, sizep)`
Returns a pointer to the next transmit buffer in the descriptor chain.
- `#define macGetNextReceiveBuffer(rdp, sizep) mac_lld_get_next_receive_buffer(rdp, sizep)`
Returns a pointer to the next receive buffer in the descriptor chain.

Configuration options

- `#define PLATFORM_MAC_USE_MAC1 FALSE`
MAC driver enable switch.

Defines

- `#define MAC_SUPPORTS_ZERO_COPY TRUE`
This implementation supports the zero-copy mode API.

Typedefs

- `typedef struct MACDriver MACDriver`
Type of a structure representing a MAC driver.

Enumerations

- `enum macstate_t { MAC_UNINIT = 0, MAC_STOP = 1, MAC_ACTIVE = 2 }`
Driver state machine possible states.

10.41.2 Function Documentation

10.41.2.1 void macInit(void)

MAC Driver initialization.

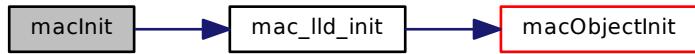
Note

This function is implicitly invoked by `halInit()`, there is no need to explicitly initialize the driver.

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:



10.41.2.2 void macObjectInit (**MACDriver** * *macp*)

Initialize the standard part of a **MACDriver** structure.

Parameters

out *macp* pointer to the **MACDriver** 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:



10.41.2.3 void macStart (**MACDriver** * *macp*, const **MACConfig** * *config*)

Configures and activates the MAC peripheral.

Parameters

in *macp* pointer to the **MACDriver** object
in *config* pointer to the **MACConfig** 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:



10.41.2.4 void macStop (MACDriver * *macp*)

Deactivates the MAC peripheral.

Parameters

in	<i>macp</i> pointer to the MACDriver 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:



10.41.2.5 msg_t macWaitTransmitDescriptor (MACDriver * *macp*, MACTransmitDescriptor * *tdp*, systime_t *time*)

Allocates a transmission descriptor.

One of the available transmission descriptors is locked and returned. If a descriptor is not currently available then the invoking thread is queued until one is freed.

Parameters

in	<i>macp</i> pointer to the MACDriver object
out	<i>tdp</i> pointer to a MACTransmitDescriptor 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

The operation status.

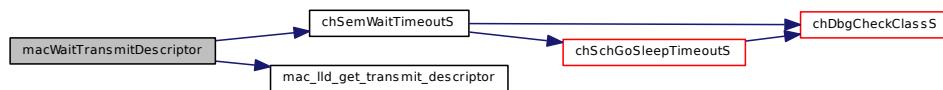
Return values

RDY_OK the descriptor was obtained.
RDY_TIMEOUT the operation timed out, descriptor not initialized.

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:

**10.41.2.6 void macReleaseTransmitDescriptor (MACTransmitDescriptor * tdp)**

Releases a transmit descriptor and starts the transmission of the enqueued data as a single frame.

Parameters

in	<i>tdp</i> the pointer to the MACTransmitDescriptor 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:

**10.41.2.7 msg_t macWaitReceiveDescriptor (MACDriver * macp, MACReceiveDescriptor * rdp, systime_t time)**

Waits for a received frame.

Stops until a frame is received and buffered. If a frame is not immediately available then the invoking thread is queued until one is received.

Parameters

in	<i>macp</i> pointer to the MACDriver object
out	<i>rdp</i> pointer to a MACReceiveDescriptor 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

The operation status.

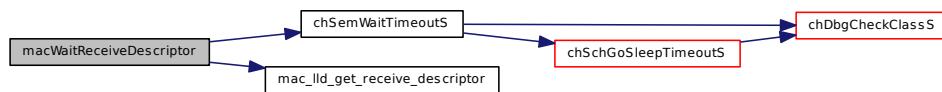
Return values

<i>RDY_OK</i>	the descriptor was obtained.
<i>RDY_TIMEOUT</i>	the operation timed out, descriptor not initialized.

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:

**10.41.2.8 void macReleaseReceiveDescriptor (**MACReceiveDescriptor** * *rdp*)**

Releases a receive descriptor.

The descriptor and its buffer are made available for more incoming frames.

Parameters

in	<i>rdp</i> the pointer to the MACReceiveDescriptor 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:

**10.41.2.9 bool_t macPollLinkStatus (**MACDriver** * *macp*)**

Updates and returns the link status.

Parameters

in	<i>macp</i> pointer to the MACDriver object
----	---

Returns

The link status.

Return values

<i>TRUE</i>	if the link is active.
<i>FALSE</i>	if the link is down.

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:

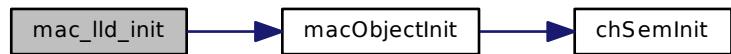
**10.41.2.10 void mac_lld_init(void)**

Low level MAC initialization.

Function Class:

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

Here is the call graph for this function:

**10.41.2.11 void mac_lld_start(MACDriver * macp)**

Configures and activates the MAC peripheral.

Parameters

in *macp* pointer to the [MACDriver](#) object

Function Class:

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

10.41.2.12 void mac_lld_stop (**MACDriver** * *macp*)

Deactivates the MAC peripheral.

Parameters

in *macp* pointer to the **MACDriver** object

Function Class:

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

10.41.2.13 msg_t mac_lld_get_transmit_descriptor (**MACDriver** * *macp*, **MACTransmitDescriptor** * *tdp*)

Returns a transmission descriptor.

One of the available transmission descriptors is locked and returned.

Parameters

in *macp* pointer to the **MACDriver** object

out *tdp* pointer to a **MACTransmitDescriptor** structure

Returns

The operation status.

Return values

RDY_OK the descriptor has been obtained.

RDY_TIMEOUT descriptor not available.

Function Class:

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

10.41.2.14 void mac_lld_release_transmit_descriptor (**MACTransmitDescriptor** * *tdp*)

Releases a transmit descriptor and starts the transmission of the enqueued data as a single frame.

Parameters

in *tdp* the pointer to the **MACTransmitDescriptor** structure

Function Class:

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

10.41.2.15 msg_t mac_lld_get_receive_descriptor (**MACDriver** * *macp*, **MACReceiveDescriptor** * *rdp*)

Returns a receive descriptor.

Parameters

in *macp* pointer to the **MACDriver** object

out *rdp* pointer to a **MACReceiveDescriptor** structure

Returns

The operation status.

Return values

RDY_OK the descriptor has been obtained.
RDY_TIMEOUT descriptor not available.

Function Class:

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

10.41.2.16 void mac_lld_release_receive_descriptor ([MACReceiveDescriptor](#) * *rdp*)

Releases a receive descriptor.

The descriptor and its buffer are made available for more incoming frames.

Parameters

in *rdp* the pointer to the [MACReceiveDescriptor](#) structure

Function Class:

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

10.41.2.17 bool_t mac_lld_poll_link_status ([MACDriver](#) * *macp*)

Updates and returns the link status.

Parameters

in *macp* pointer to the [MACDriver](#) object

Returns

The link status.

Return values

TRUE if the link is active.
FALSE if the link is down.

Function Class:

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

10.41.2.18 size_t mac_lld_write_transmit_descriptor ([MACTransmitDescriptor](#) * *tdp*, uint8_t * *buf*, size_t *size*)

Writes to a transmit descriptor's stream.

Parameters

in *tdp* pointer to a [MACTransmitDescriptor](#) structure
in *buf* pointer to the buffer containing the data to be written
in *size* number of bytes to be written

Returns

The number of bytes written into the descriptor's stream, this value can be less than the amount specified in the parameter *size* if the maximum frame size is reached.

Function Class:

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

10.41.2.19 size_t mac_lld_read_receive_descriptor (**MACReceiveDescriptor * *rdp*, **uint8_t** * *buf*, **size_t** *size*)**

Reads from a receive descriptor's stream.

Parameters

in	<i>rdp</i>	pointer to a MACReceiveDescriptor structure
in	<i>buf</i>	pointer to the buffer that will receive the read data
in	<i>size</i>	number of bytes to be read

Returns

The number of bytes read from the descriptor's stream, this value can be less than the amount specified in the parameter *size* if there are no more bytes to read.

Function Class:

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

10.41.2.20 uint8_t* mac_lld_get_next_transmit_buffer (**MACTransmitDescriptor * *tdp*, **size_t** *size*, **size_t** * *sizep*)**

Returns a pointer to the next transmit buffer in the descriptor chain.

Note

The API guarantees that enough buffers can be requested to fill a whole frame.

Parameters

in	<i>tdp</i>	pointer to a MACTransmitDescriptor structure
in	<i>size</i>	size of the requested buffer. Specify the frame size on the first call then scale the value down subtracting the amount of data already copied into the previous buffers.
out	<i>sizep</i>	pointer to variable receiving the buffer size, it is zero when the last buffer has already been returned. Note that a returned size lower than the amount requested means that more buffers must be requested in order to fill the frame data entirely.

Returns

Pointer to the returned buffer.

Return values

NULL if the buffer chain has been entirely scanned.

Function Class:

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

10.41.2.21 const uint8_t* mac_lld_get_next_receive_buffer (**MACReceiveDescriptor * *rdp*, **size_t** * *sizep*)**

Returns a pointer to the next receive buffer in the descriptor chain.

Note

The API guarantees that the descriptor chain contains a whole frame.

Parameters

in	<i>rdp</i>	pointer to a MACReceiveDescriptor structure
out	<i>sizep</i>	pointer to variable receiving the buffer size, it is zero when the last buffer has already been returned.

Returns

Pointer to the returned buffer.

Return values

NULL if the buffer chain has been entirely scanned.

Function Class:

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

10.41.3 Variable Documentation**10.41.3.1 MACDriver ETHD1**

MAC1 driver identifier.

10.41.4 Define Documentation**10.41.4.1 #define MAC_USE_ZERO_COPY FALSE**

Enables an event sources for incoming packets.

10.41.4.2 #define MAC_USE_EVENTS TRUE

Enables an event sources for incoming packets.

10.41.4.3 #define macGetReceiveEventSource(*macp*)(&(*macp*)>rdevent)

Returns the received frames event source.

Parameters

in	<i>macp</i>	pointer to the MACDriver object
----	-------------	---

Returns

The pointer to the [EventSource](#) structure.

Function Class:

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

10.41.4.4 #define macWriteTransmitDescriptor(*tdp*, *buf*, *size*) mac_lld_write_transmit_descriptor(*tdp*, *buf*, *size*)

Writes to a transmit descriptor's stream.

Parameters

in	<i>tdp</i>	pointer to a MACTransmitDescriptor structure
in	<i>buf</i>	pointer to the buffer containing the data to be written
in	<i>size</i>	number of bytes to be written

Returns

The number of bytes written into the descriptor's stream, this value can be less than the amount specified in the parameter `size` if the maximum frame size is reached.

Function Class:

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

```
10.41.4.5 #define macReadReceiveDescriptor( rdp, buf, size ) mac_lld.read_receive_descriptor(rdp, buf, size)
```

Reads from a receive descriptor's stream.

Parameters

in	<code>rdp</code>	pointer to a <code>MACReceiveDescriptor</code> structure
in	<code>buf</code>	pointer to the buffer that will receive the read data
in	<code>size</code>	number of bytes to be read

Returns

The number of bytes read from the descriptor's stream, this value can be less than the amount specified in the parameter `size` if there are no more bytes to read.

Function Class:

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

```
10.41.4.6 #define macGetNextTransmitBuffer( tdp, size, sizep ) mac_lld.get.next.transmit.buffer(tdp, size, sizep)
```

Returns a pointer to the next transmit buffer in the descriptor chain.

Note

The API guarantees that enough buffers can be requested to fill a whole frame.

Parameters

in	<code>tdp</code>	pointer to a <code>MACTransmitDescriptor</code> structure
in	<code>size</code>	size of the requested buffer. Specify the frame size on the first call then scale the value down subtracting the amount of data already copied into the previous buffers.
out	<code>sizep</code>	pointer to variable receiving the real buffer size. The returned value can be less than the amount requested, this means that more buffers must be requested in order to fill the frame data entirely.

Returns

Pointer to the returned buffer.

Function Class:

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

```
10.41.4.7 #define macGetNextReceiveBuffer( rdp, sizep ) mac_lld.get.next.receive.buffer(rdp, sizep)
```

Returns a pointer to the next receive buffer in the descriptor chain.

Note

The API guarantees that the descriptor chain contains a whole frame.

Parameters

in	<i>rdp</i>	pointer to a <code>MACReceiveDescriptor</code> structure
out	<i>sizep</i>	pointer to variable receiving the buffer size, it is zero when the last buffer has already been returned.

Returns

Pointer to the returned buffer.

Return values

`NULL` if the buffer chain has been entirely scanned.

Function Class:

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

10.41.4.8 #define MAC_SUPPORTS_ZERO_COPY TRUE

This implementation supports the zero-copy mode API.

10.41.4.9 #define PLATFORM_MAC_USE_MAC1 FALSE

MAC driver enable switch.

If set to `TRUE` the support for MAC1 is included.

10.41.5 Typedef Documentation**10.41.5.1 typedef struct MACDriver MACDriver**

Type of a structure representing a MAC driver.

10.41.6 Enumeration Type Documentation**10.41.6.1 enum macstate_t**

Driver state machine possible states.

Enumerator:

`MAC_UNINIT` Not initialized.

`MAC_STOP` Stopped.

`MAC_ACTIVE` Active.

10.42 MMC over SPI Driver**10.42.1 Detailed Description**

Generic MMC driver. This module implements a portable MMC/SD driver that uses a SPI driver as physical layer. Hot plugging and removal are supported through kernel events.

Precondition

In order to use the `MMC_SPI` driver the `HAL_USE_MMC_SPI` and `HAL_USE_SPI` options must be enabled in `halconf.h`.

10.42.2 Driver State Machine

This driver implements a state machine internally, see the [Abstract I/O Block Device](#) module documentation for details.

10.42.3 Driver Operations

This driver allows to read or write single or multiple 512 bytes blocks on a SD Card.

Data Structures

- struct [MMCConfig](#)
MMC/SD over SPI driver configuration structure.
- struct [MMCDriverVMT](#)
MMCDriver virtual methods table.
- struct [MMCDriver](#)
Structure representing a MMC/SD over SPI driver.

Functions

- void [mmcInit](#) (void)
MMC over SPI driver initialization.
- void [mmcObjectInit](#) ([MMCDriver](#) *mmcp)
Initializes an instance.
- void [mmcStart](#) ([MMCDriver](#) *mmcp, const [MMCConfig](#) *config)
Configures and activates the MMC peripheral.
- void [mmcStop](#) ([MMCDriver](#) *mmcp)
Disables the MMC peripheral.
- [bool_t](#) [mmcConnect](#) ([MMCDriver](#) *mmcp)
Performs the initialization procedure on the inserted card.
- [bool_t](#) [mmcDisconnect](#) ([MMCDriver](#) *mmcp)
Brings the driver in a state safe for card removal.
- [bool_t](#) [mmcStartSequentialRead](#) ([MMCDriver](#) *mmcp, [uint32_t](#) startblk)
Starts a sequential read.
- [bool_t](#) [mmcSequentialRead](#) ([MMCDriver](#) *mmcp, [uint8_t](#) *buffer)
Reads a block within a sequential read operation.
- [bool_t](#) [mmcStopSequentialRead](#) ([MMCDriver](#) *mmcp)
Stops a sequential read gracefully.
- [bool_t](#) [mmcStartSequentialWrite](#) ([MMCDriver](#) *mmcp, [uint32_t](#) startblk)
Starts a sequential write.
- [bool_t](#) [mmcSequentialWrite](#) ([MMCDriver](#) *mmcp, const [uint8_t](#) *buffer)
Writes a block within a sequential write operation.
- [bool_t](#) [mmcStopSequentialWrite](#) ([MMCDriver](#) *mmcp)
Stops a sequential write gracefully.
- [bool_t](#) [mmcSync](#) ([MMCDriver](#) *mmcp)
Waits for card idle condition.
- [bool_t](#) [mmcGetInfo](#) ([MMCDriver](#) *mmcp, [BlockDeviceInfo](#) *bdip)
Returns the media info.
- [bool_t](#) [mmcErase](#) ([MMCDriver](#) *mmcp, [uint32_t](#) startblk, [uint32_t](#) endblk)
Erases blocks.

MMC_SPI configuration options

- #define `MMC_NICE_WAITING` TRUE
Delays insertions.

Macro Functions

- #define `mmclsCardInserted`(`mmcp`) `mmc_lld_is_card_inserted`(`mmcp`)
Returns the card insertion status.
- #define `mmclsWriteProtected`(`mmcp`) `mmc_lld_is_write_protected`(`mmcp`)
Returns the write protect status.

Defines

- #define `_mmc_driver_methods` `_mmcsd_block_device_methods`
MMCDriver specific methods.

10.42.4 Function Documentation

10.42.4.1 void mmcInit(void)

MMC over SPI driver initialization.

Note

This function is implicitly invoked by `halInit()`, there is no need to explicitly initialize the driver.

Function Class:

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

10.42.4.2 void mmcObjectInit(MMCDriver * *mmcp*)

Initializes an instance.

Parameters

out	<code>mmcp</code> pointer to the <code>MMCDriver</code> object
-----	--

Function Class:

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

10.42.4.3 void mmcStart(MMCDriver * *mmcp*, const MMCConfig * *config*)

Configures and activates the MMC peripheral.

Parameters

in	<code>mmcp</code> pointer to the <code>MMCDriver</code> object
in	<code>config</code> pointer to the <code>MMCConfig</code> object.

Function Class:

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

10.42.4.4 void mmcStop (**MMCDriver** * *mmcp*)

Disables the MMC peripheral.

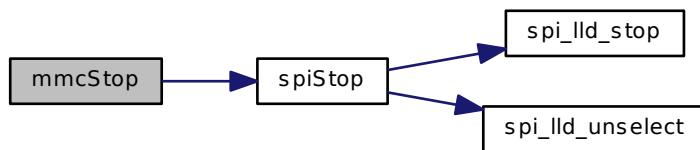
Parameters

in *mmcp* pointer to the **MMCDriver** 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:



10.42.4.5 **bool_t** mmcConnect (**MMCDriver** * *mmcp*)

Performs the initialization procedure on the inserted card.

This function should be invoked when a card is inserted and brings the driver in the **MMC_READY** state where it is possible to perform read and write operations.

Note

It is possible to invoke this function from the insertion event handler.

Parameters

in *mmcp* pointer to the **MMCDriver** object

Returns

The operation status.

Return values

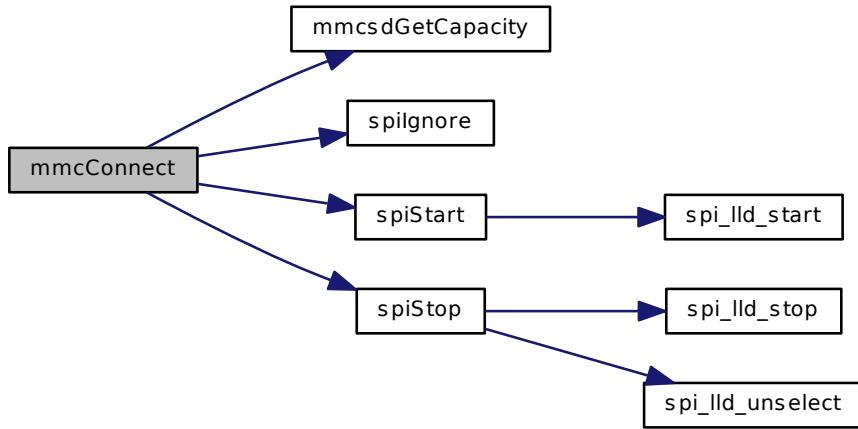
CH_SUCCESS the operation succeeded and the driver is now in the **MMC_READY** state.

CH_FAILED the operation 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:



10.42.4.6 `bool_t mmcDisconnect (MMCDriver * mmcp)`

Brings the driver in a state safe for card removal.

Parameters

`in mmcp` pointer to the `MMCDriver` object

Returns

The operation status.

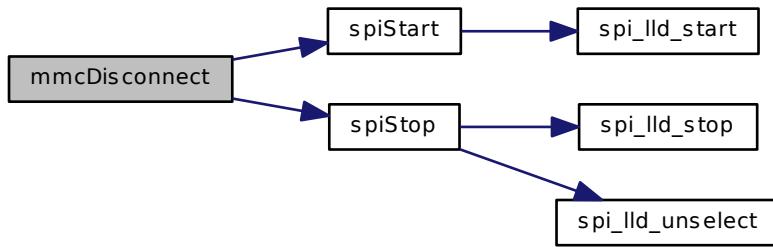
Return values

<code>CH_SUCCESS</code>	the operation succeeded and the driver is now in the <code>MMC_INSERTED</code> state.
<code>CH_FAILED</code>	the operation 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:



10.42.4.7 `bool_t mmcStartSequentialRead (MMCDriver * mmcp, uint32_t startblk)`

Starts a sequential read.

Parameters

in	<code>mmcp</code>	pointer to the <code>MMCDriver</code> object
in	<code>startblk</code>	first block to read

Returns

The operation status.

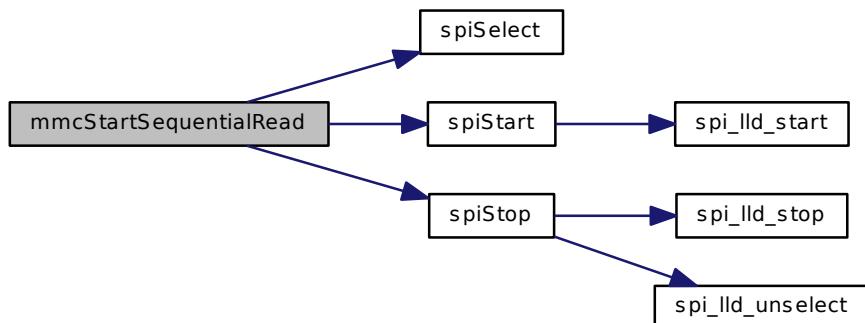
Return values

<code>CH_SUCCESS</code>	the operation succeeded.
<code>CH_FAILED</code>	the operation 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:



10.42.4.8 `bool_t mmcSequentialRead(MMCDriver * mmcp, uint8_t * buffer)`

Reads a block within a sequential read operation.

Parameters

in	<i>mmcp</i>	pointer to the <code>MMCDriver</code> object
out	<i>buffer</i>	pointer to the read buffer

Returns

The operation status.

Return values

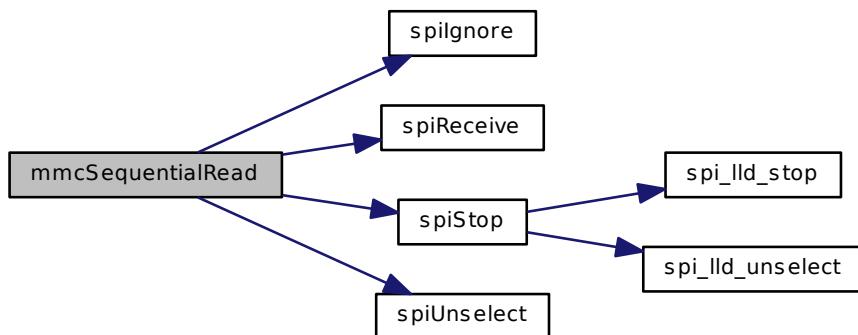
`CH_SUCCESS` the operation succeeded.

`CH_FAILED` the operation 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:



10.42.4.9 `bool_t mmcStopSequentialRead(MMCDriver * mmcp)`

Stops a sequential read gracefully.

Parameters

in	<i>mmcp</i>	pointer to the <code>MMCDriver</code> object
----	-------------	--

Returns

The operation status.

Return values

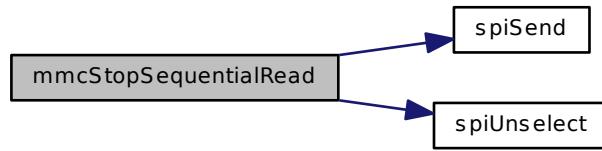
`CH_SUCCESS` the operation succeeded.

`CH_FAILED` the operation 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:

**10.42.4.10 bool_t mmcStartSequentialWrite (MMCDriver * *mmcp*, uint32_t *startblk*)**

Starts a sequential write.

Parameters

in	<i>mmcp</i>	pointer to the MMCDriver object
in	<i>startblk</i>	first block to write

Returns

The operation status.

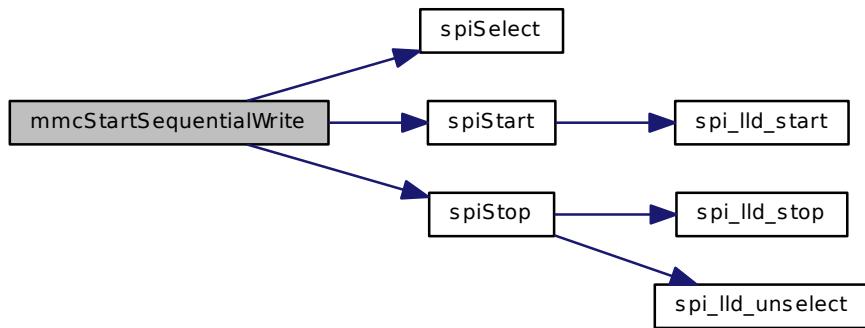
Return values

<i>CH_SUCCESS</i>	the operation succeeded.
<i>CH_FAILED</i>	the operation 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:



10.42.4.11 `bool_t mmcSequentialWrite(MMCDriver * mmcp, const uint8_t * buffer)`

Writes a block within a sequential write operation.

Parameters

in	<code>mmcp</code>	pointer to the <code>MMCDriver</code> object
out	<code>buffer</code>	pointer to the write buffer

Returns

The operation status.

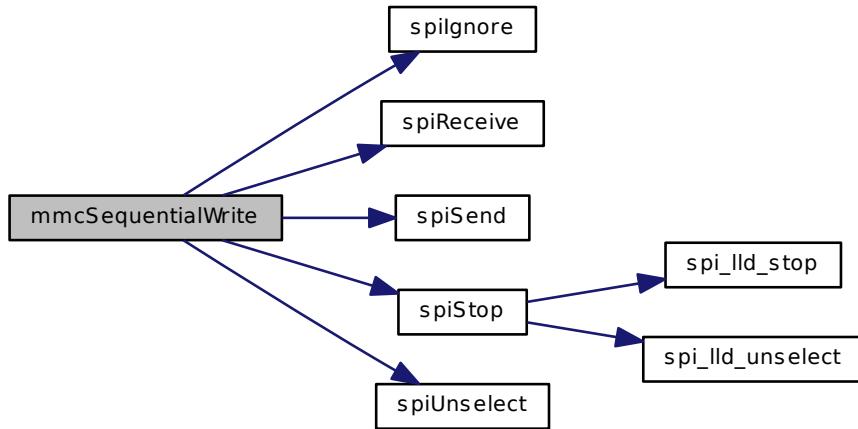
Return values

<code>CH_SUCCESS</code>	the operation succeeded.
<code>CH_FAILED</code>	the operation 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:



10.42.4.12 bool_t mmcStopSequentialWrite (MMCDriver * mmcp)

Stops a sequential write gracefully.

Parameters

in *mmcp* pointer to the [MMCDriver](#) object

Returns

The operation status.

Return values

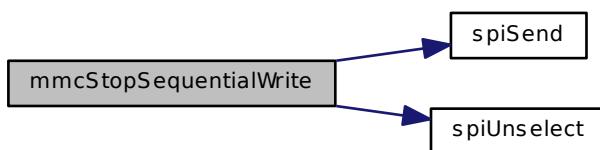
CH_SUCCESS the operation succeeded.

CH_FAILED the operation 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:



10.42.4.13 `bool_t mmcSync (MMCDriver * mmcp)`

Waits for card idle condition.

Parameters

in	<code>mmcp</code> pointer to the <code>MMCDriver</code> object
----	--

Returns

The operation status.

Return values

`CH_SUCCESS` the operation succeeded.

`CH_FAILED` the operation 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:



10.42.4.14 `bool_t mmcGetInfo (MMCDriver * mmcp, BlockDeviceInfo * bdip)`

Returns the media info.

Parameters

in	<code>mmcp</code> pointer to the <code>MMCDriver</code> object
out	<code>bdip</code> pointer to a <code>BlockDeviceInfo</code> structure

Returns

The operation status.

Return values

`CH_SUCCESS` the operation succeeded.

`CH_FAILED` the operation failed.

Function Class:

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

10.42.4.15 `bool_t mmcErase (MMCDriver * mmcp, uint32_t startblk, uint32_t endblk)`

Erases blocks.

Parameters

in	<i>mmcp</i>	pointer to the <code>MMCDriver</code> object
in	<i>startblk</i>	starting block number
in	<i>endblk</i>	ending block number

Returns

The operation status.

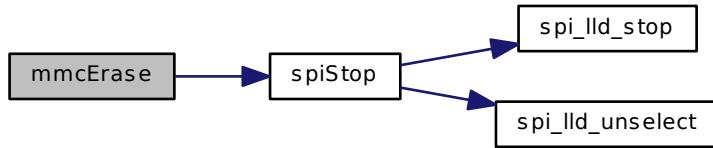
Return values

<i>CH_SUCCESS</i>	the operation succeeded.
<i>CH_FAILED</i>	the operation 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:

**10.42.5 Define Documentation****10.42.5.1 #define MMC_NICE_WAITING TRUE**

Delays insertions.

If enabled this option inserts delays into the MMC waiting routines releasing some extra CPU time for the threads with lower priority, this may slow down the driver a bit however. This option is recommended also if the SPI driver does not use a DMA channel and heavily loads the CPU.

10.42.5.2 #define _mmc_driver_methods _mmcsd_block_device_methods

`MMCDriver` specific methods.

10.42.5.3 #define mmclsCardInserted(*mmcp*) mmc_lld_is_card_inserted(*mmcp*)

Returns the card insertion status.

Note

This macro wraps a low level function named `sdc_lld_is_card_inserted()`, this function must be provided by the application because it is not part of the SDC driver.

Parameters

in	<i>mmcp</i>	pointer to the <code>MMCDriver</code> object
----	-------------	--

Returns

The card state.

Return values

<i>FALSE</i>	card not inserted.
<i>TRUE</i>	card inserted.

Function Class:

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

10.42.5.4 #define mmclsWriteProtected(*mmcp*) mmc_lld_is_write_protected(*mmcp*)

Returns the write protect status.

Parameters

in *mmcp* pointer to the [MMCDriver](#) object

Returns

The card state.

Return values

<i>FALSE</i>	card not inserted.
<i>TRUE</i>	card inserted.

Function Class:

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

10.43 MMC/SD Block Device

10.43.1 Detailed Description

This module implements a common ancestor for all device drivers accessing MMC or SD cards. This interface inherits the state machine and the interface from the [Abstract I/O Block Device](#) module.

Data Structures

- struct [MMCSDBlockDeviceVMT](#)
MMCSDBlockDevice virtual methods table.
- struct [MMCSDBlockDevice](#)
MCC/SD block device class.

Functions

- uint32_t [mmcsdGetCapacity](#) (uint32_t csd[4])
Extract card capacity from a CSD.

SD/MMC status conditions

- #define [MMCSD_STS_IDLE](#) 0

- #define MMCSD_STS_READY 1
- #define MMCSD_STS_IDENT 2
- #define MMCSD_STS_STBY 3
- #define MMCSD_STS_TRAN 4
- #define MMCSD_STS_DATA 5
- #define MMCSD_STS_RCV 6
- #define MMCSD_STS_PRG 7
- #define MMCSD_STS_DIS 8

SD/MMC commands

- #define MMCSD_CMD_GO_IDLE_STATE 0
- #define MMCSD_CMD_INIT 1
- #define MMCSD_CMD_ALL_SEND_CID 2
- #define MMCSD_CMD_SEND_RELATIVE_ADDR 3
- #define MMCSD_CMD_SET_BUS_WIDTH 6
- #define MMCSD_CMD_SEL_DESEL_CARD 7
- #define MMCSD_CMD_SEND_IF_COND 8
- #define MMCSD_CMD_SEND_CSD 9
- #define MMCSD_CMD_SEND_CID 10
- #define MMCSD_CMD_STOP_TRANSMISSION 12
- #define MMCSD_CMD_SEND_STATUS 13
- #define MMCSD_CMD_SET_BLOCKLEN 16
- #define MMCSD_CMD_READ_SINGLE_BLOCK 17
- #define MMCSD_CMD_READ_MULTIPLE_BLOCK 18
- #define MMCSD_CMD_SET_BLOCK_COUNT 23
- #define MMCSD_CMD_WRITE_BLOCK 24
- #define MMCSD_CMD_WRITE_MULTIPLE_BLOCK 25
- #define MMCSD_CMD_ERASE_RW_BLK_START 32
- #define MMCSD_CMD_ERASE_RW_BLK_END 33
- #define MMCSD_CMD_ERASE 38
- #define MMCSD_CMD_APP_OP_COND 41
- #define MMCSD_CMD_LOCK_UNLOCK 42
- #define MMCSD_CMD_APP_CMD 55
- #define MMCSD_CMD_READ_OCR 58

CSD record offsets

- #define MMCSD_CSD_20_CRC_SLICE 7,1
Slice position of values in CSD register.
- #define MMCSD_CSD_20_FILE_FORMAT_SLICE 11,10
- #define MMCSD_CSD_20_TMP_WRITE_PROTECT_SLICE 12,12
- #define MMCSD_CSD_20_PERM_WRITE_PROTECT_SLICE 13,13
- #define MMCSD_CSD_20_COPY_SLICE 14,14
- #define MMCSD_CSD_20_FILE_FORMAT_GRP_SLICE 15,15
- #define MMCSD_CSD_20_WRITE_BL_PARTIAL_SLICE 21,21
- #define MMCSD_CSD_20_WRITE_BL_LEN_SLICE 25,12
- #define MMCSD_CSD_20_R2W_FACTOR_SLICE 28,26
- #define MMCSD_CSD_20_WP_GRP_ENABLE_SLICE 31,31
- #define MMCSD_CSD_20_WP_GRP_SIZE_SLICE 38,32
- #define MMCSD_CSD_20_ERASE_SECTOR_SIZE_SLICE 45,39
- #define MMCSD_CSD_20_ERASE_BLK_EN_SLICE 46,46
- #define MMCSD_CSD_20_C_SIZE_SLICE 69,48
- #define MMCSD_CSD_20_DSR_IMP_SLICE 76,76

- #define **MMCSD_CSD_20_READ_BLK_MISALIGN_SLICE** 77,77
- #define **MMCSD_CSD_20_WRITE_BLK_MISALIGN_SLICE** 78,78
- #define **MMCSD_CSD_20_READ_BL_PARTIAL_SLICE** 79,79
- #define **MMCSD_CSD_20_READ_BL_LEN_SLICE** 83,80
- #define **MMCSD_CSD_20_CCC_SLICE** 95,84
- #define **MMCSD_CSD_20_TRANS_SPEED_SLICE** 103,96
- #define **MMCSD_CSD_20_NSAC_SLICE** 111,104
- #define **MMCSD_CSD_20_TAAC_SLICE** 119,112
- #define **MMCSD_CSD_20_STRUCTURE_SLICE** 127,126
- #define **MMCSD_CSD_10_CRC_SLICE** MMCSD_CSD_20_CRC_SLICE
- #define **MMCSD_CSD_10_FILE_FORMAT_SLICE** MMCSD_CSD_20_FILE_FORMAT_SLICE
- #define **MMCSD_CSD_10_TMP_WRITE_PROTECT_SLICE** MMCSD_CSD_20_TMP_WRITE_PROTECT_SLICE
- #define **MMCSD_CSD_10_PERM_WRITE_PROTECT_SLICE** MMCSD_CSD_20_PERM_WRITE_PROTECT_SLICE
- #define **MMCSD_CSD_10_COPY_SLICE** MMCSD_CSD_20_COPY_SLICE
- #define **MMCSD_CSD_10_FILE_FORMAT_GRP_SLICE** MMCSD_CSD_20_FILE_FORMAT_GRP_SLICE
- #define **MMCSD_CSD_10_WRITE_BL_PARTIAL_SLICE** MMCSD_CSD_20_WRITE_BL_PARTIAL_SLICE
- #define **MMCSD_CSD_10_WRITE_BL_LEN_SLICE** MMCSD_CSD_20_WRITE_BL_LEN_SLICE
- #define **MMCSD_CSD_10_R2W_FACTOR_SLICE** MMCSD_CSD_20_R2W_FACTOR_SLICE
- #define **MMCSD_CSD_10_WP_GRP_ENABLE_SLICE** MMCSD_CSD_20_WP_GRP_ENABLE_SLICE
- #define **MMCSD_CSD_10_WP_GRP_SIZE_SLICE** MMCSD_CSD_20_WP_GRP_SIZE_SLICE
- #define **MMCSD_CSD_10_ERASE_SECTOR_SIZE_SLICE** MMCSD_CSD_20_ERASE_SECTOR_SIZE_SLICE
- #define **MMCSD_CSD_10_ERASE_BLK_EN_SLICE** MMCSD_CSD_20_ERASE_BLK_EN_SLICE
- #define **MMCSD_CSD_10_C_SIZE_MULT_SLICE** 49,47
- #define **MMCSD_CSD_10_VDD_W_CURR_MAX_SLICE** 52,50
- #define **MMCSD_CSD_10_VDD_W_CURR_MIN_SLICE** 55,53
- #define **MMCSD_CSD_10_VDD_R_CURR_MAX_SLICE** 58,56
- #define **MMCSD_CSD_10_VDD_R_CURR_MIX_SLICE** 61,59
- #define **MMCSD_CSD_10_C_SIZE_SLICE** 73,62
- #define **MMCSD_CSD_10_DSR_IMP_SLICE** MMCSD_CSD_20_DSR_IMP_SLICE
- #define **MMCSD_CSD_10_READ_BLK_MISALIGN_SLICE** MMCSD_CSD_20_READ_BLK_MISALIGN_SLICE
- #define **MMCSD_CSD_10_WRITE_BLK_MISALIGN_SLICE** MMCSD_CSD_20_WRITE_BLK_MISALIGN_SLICE
- #define **MMCSD_CSD_10_READ_BL_PARTIAL_SLICE** MMCSD_CSD_20_READ_BL_PARTIAL_SLICE
- #define **MMCSD_CSD_10_READ_BL_LEN_SLICE** 83,80
- #define **MMCSD_CSD_10_CCC_SLICE** MMCSD_CSD_20_CCC_SLICE
- #define **MMCSD_CSD_10_TRANS_SPEED_SLICE** MMCSD_CSD_20_TRANS_SPEED_SLICE
- #define **MMCSD_CSD_10_NSAC_SLICE** MMCSD_CSD_20_NSAC_SLICE
- #define **MMCSD_CSD_10_TAAC_SLICE** MMCSD_CSD_20_TAAC_SLICE
- #define **MMCSD_CSD_10_STRUCTURE_SLICE** MMCSD_CSD_20_STRUCTURE_SLICE

R1 response utilities

- #define **MMCSD_R1_ERROR**(r1) (((r1) & MMCSD_R1_ERROR_MASK) != 0)

Evaluates to TRUE if the R1 response contains error flags.
- #define **MMCSD_R1_STS**(r1) (((r1) >> 9) & 15)

Returns the status field of an R1 response.
- #define **MMCSD_R1_IS_CARD_LOCKED**(r1) (((r1) >> 21) & 1)

Evaluates to TRUE if the R1 response indicates a locked card.

Macro Functions

- `#define mmcsdGetCardCapacity(ip) ((ip)->capacity)`
Returns the card capacity in blocks.

Defines

- `#define MMCSD_BLOCK_SIZE 512`
Fixed block size for MMC/SD block devices.
- `#define MMCSD_R1_ERROR_MASK 0xFDFFFE008`
Mask of error bits in R1 responses.
- `#define MMCSD_CMD8_PATTERN 0x000001AA`
Fixed pattern for CMD8.
- `#define _mmcsd_block_device_methods _base_block_device_methods`
MMCSD block device specific methods.
- `#define _mmcsd_block_device_data`
MMCSD block device specific data.

10.43.2 Function Documentation

10.43.2.1 `uint32_t mmcsdGetCapacity (uint32_t csd[4])`

Extract card capacity from a CSD.

The capacity is returned as number of available blocks.

Parameters

in `csd` the CSD record

Returns

The card capacity.

Return values

0 CSD format error

10.43.3 Define Documentation

10.43.3.1 `#define MMCSD_BLOCK_SIZE 512`

Fixed block size for MMC/SD block devices.

10.43.3.2 `#define MMCSD_R1_ERROR_MASK 0xFDFFFE008`

Mask of error bits in R1 responses.

10.43.3.3 `#define MMCSD_CMD8_PATTERN 0x000001AA`

Fixed pattern for CMD8.

10.43.3.4 #define MMCSD_CSD_20_CRC_SLICE 7,1

Slice position of values in CSD register.

10.43.3.5 #define _mmcsd_block_device_methods _base_block_device_methods

[MMCSDBlockDevice](#) specific methods.

10.43.3.6 #define _mmcsd_block_device_data

Value:

```
_base_block_device_data
/* Card CID.*/
uint32_t cid[4];
/* Card CSD.*/
uint32_t csd[4];
/* Total number of blocks in card.*/
uint32_t capacity;
```

[MMCSDBlockDevice](#) specific data.

Note

It is empty because [MMCSDBlockDevice](#) is only an interface without implementation.

10.43.3.7 #define MMCSD_R1_ERROR(r1) (((r1) & MMCSD_R1_ERROR_MASK) != 0)

Evaluates to TRUE if the R1 response contains error flags.

Parameters

in	r1 the r1 response
----	--------------------

10.43.3.8 #define MMCSD_R1_STS(r1) (((r1) >> 9) & 15)

Returns the status field of an R1 response.

Parameters

in	r1 the r1 response
----	--------------------

10.43.3.9 #define MMCSD_R1_IS_CARD_LOCKED(r1) (((r1) >> 21) & 1)

Evaluates to TRUE if the R1 response indicates a locked card.

Parameters

in	r1 the r1 response
----	--------------------

10.43.3.10 #define mmcsdGetCardCapacity(ip) ((ip)->capacity)

Returns the card capacity in blocks.

Parameters

in *ip* pointer to a [MMCSDBlockDevice](#) or derived class

Returns

The card capacity.

Function Class:

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

10.44 PAL Driver

10.44.1 Detailed Description

I/O Ports Abstraction Layer. This module defines an abstract interface for digital I/O ports. Note that most I/O ports functions are just macros. The macros have default software implementations that can be redefined in a PAL Low Level Driver if the target hardware supports special features like, for example, atomic bit set/reset/masking. Please refer to the ports specific documentation for details.

The [PAL Driver](#) has the advantage to make the access to the I/O ports platform independent and still be optimized for the specific architectures.

Note that the PAL Low Level Driver may also offer non standard macro and functions in order to support specific features but, of course, the use of such interfaces would not be portable. Such interfaces shall be marked with the architecture name inside the function names.

Precondition

In order to use the PAL driver the `HAL_USE_PAL` option must be enabled in `halconf.h`.

10.44.2 Implementation Rules

In implementing a PAL Low Level Driver there are some rules/behaviors that should be respected.

10.44.2.1 Writing on input pads

The behavior is not specified but there are implementations better than others, this is the list of possible implementations, preferred options are on top:

1. The written value is not actually output but latched, should the pads be reprogrammed as outputs the value would be in effect.
2. The write operation is ignored.
3. The write operation has side effects, as example disabling/enabling pull up/down resistors or changing the pad direction. This scenario is discouraged, please try to avoid this scenario.

10.44.2.2 Reading from output pads

The behavior is not specified but there are implementations better than others, this is the list of possible implementations, preferred options are on top:

1. The actual pads states are read (not the output latch).
2. The output latch value is read (regardless of the actual pads states).
3. Unspecified, please try to avoid this scenario.

10.44.2.3 Writing unused or unimplemented port bits

The behavior is not specified.

10.44.2.4 Reading from unused or unimplemented port bits

The behavior is not specified.

10.44.2.5 Reading or writing on pins associated to other functionalities

The behavior is not specified.

Data Structures

- struct `IOBus`
I/O bus descriptor.
- struct `PALConfig`
Generic I/O ports static initializer.

Functions

- `ioportmask_t palReadBus (IOBus *bus)`
Read from an I/O bus.
- `void palWriteBus (IOBus *bus, ioportmask_t bits)`
Write to an I/O bus.
- `void palSetBusMode (IOBus *bus, iomode_t mode)`
Programs a bus with the specified mode.
- `void _pal_lld_init (const PALConfig *config)`
STM32 I/O ports configuration.
- `void _pal_lld_setgroupmode (ioportid_t port, ioportmask_t mask, iomode_t mode)`
Pads mode setup.

Pads mode constants

- `#define PAL_MODE_RESET 0`
After reset state.
- `#define PAL_MODE_UNCONNECTED 1`
*Safe state for **unconnected** pads.*
- `#define PAL_MODE_INPUT 2`
Regular input high-Z pad.
- `#define PAL_MODE_INPUT_PULLUP 3`
Input pad with weak pull up resistor.
- `#define PAL_MODE_INPUT_PULLDOWN 4`
Input pad with weak pull down resistor.
- `#define PAL_MODE_INPUT_ANALOG 5`
Analog input mode.
- `#define PAL_MODE_OUTPUT_PUSH_PULL 6`
Push-pull output pad.
- `#define PAL_MODE_OUTPUT_OPENDRAIN 7`
Open-drain output pad.

Logic level constants

- `#define PAL_LOW 0`

Logical low state.

- `#define PAL_HIGH 1`

Logical high state.

Macro Functions

- `#define palInit(config) pal_lld_init(config)`

PAL subsystem initialization.

- `#define palReadPort(port) ((void)(port), 0)`

Reads the physical I/O port states.

- `#define palReadLatch(port) ((void)(port), 0)`

Reads the output latch.

- `#define palWritePort(port, bits) ((void)(port), (void)(bits))`

Writes a bits mask on a I/O port.

- `#define palSetPort(port, bits) palWritePort(port, palReadLatch(port) | (bits))`

Sets a bits mask on a I/O port.

- `#define palClearPort(port, bits) palWritePort(port, palReadLatch(port) & ~ (bits))`

Clears a bits mask on a I/O port.

- `#define palTogglePort(port, bits) palWritePort(port, palReadLatch(port) ^ (bits))`

Toggles a bits mask on a I/O port.

- `#define palReadGroup(port, mask, offset) ((palReadPort(port) >> (offset)) & (mask))`

Reads a group of bits.

- `#define palWriteGroup(port, mask, offset, bits)`

Writes a group of bits.

- `#define palSetGroupMode(port, mask, offset, mode)`

Pads group mode setup.

- `#define palReadPad(port, pad) ((palReadPort(port) >> (pad)) & 1)`

Reads an input pad logical state.

- `#define palWritePad(port, pad, bit)`

Writes a logical state on an output pad.

- `#define palSetPad(port, pad) palSetPort(port, PAL_PORT_BIT(pad))`

Sets a pad logical state to `PAL_HIGH`.

- `#define palClearPad(port, pad) palClearPort(port, PAL_PORT_BIT(pad))`

Clears a pad logical state to `PAL_LOW`.

- `#define palTogglePad(port, pad) palTogglePort(port, PAL_PORT_BIT(pad))`

Toggles a pad logical state.

- `#define palSetPadMode(port, pad, mode) palSetGroupMode(port, PAL_PORT_BIT(pad), 0, mode)`

Pad mode setup.

Defines

- `#define PAL_PORT_BIT(n) ((ioportmask_t)(1 << (n)))`

Port bit helper macro.

- `#define PAL_GROUP_MASK(width) ((ioportmask_t)(1 << (width)) - 1)`

Bits group mask helper.

- `#define _IOBUS_DATA(name, port, width, offset) {port, PAL_GROUP_MASK(width), offset}`

Data part of a static I/O bus initializer.

- `#define IOBUS_DECL(name, port, width, offset) IOBus name = _IOBUS_DATA(name, port, width, offset)`

- **#define PAL_IOPORTS_WIDTH 32**
Width, in bits, of an I/O port.
- **#define PAL_WHOLE_PORT ((ioportmask_t)0xFFFFFFFF)**
Whole port mask.
- **#define IOPORT1 0**
First I/O port identifier.
- **#define pal_lld_init(config) _pal_lld_init(config)**
Low level PAL subsystem initialization.
- **#define pal_lld_readport(port) 0**
Reads the physical I/O port states.
- **#define pal_lld_readlatch(port) 0**
Reads the output latch.
- **#define pal_lld_writeport(port, bits)**
Writes a bits mask on a I/O port.
- **#define pal_lld_setport(port, bits)**
Sets a bits mask on a I/O port.
- **#define pal_lld_clearport(port, bits)**
Clears a bits mask on a I/O port.
- **#define pal_lld_toggleport(port, bits)**
Toggles a bits mask on a I/O port.
- **#define pal_lld_readgroup(port, mask, offset) 0**
Reads a group of bits.
- **#define pal_lld_writegroup(port, mask, offset, bits) (void)bits**
Writes a group of bits.
- **#define pal_lld_setgroupmode(port, mask, offset, mode) _pal_lld_setgroupmode(port, mask << offset, mode)**
Pads group mode setup.
- **#define pal_lld_readpad(port, pad) PAL_LOW**
Reads a logical state from an I/O pad.
- **#define pal_lld_writepad(port, pad, bit)**
Writes a logical state on an output pad.
- **#define pal_lld_setpad(port, pad)**
Sets a pad logical state to PAL_HIGH.
- **#define pal_lld_clearpad(port, pad)**
Clears a pad logical state to PAL_LOW.
- **#define pal_lld_togglepad(port, pad)**
Toggles a pad logical state.
- **#define pal_lld_setpadmode(port, pad, mode)**
Pad mode setup.

Typedefs

- **typedef uint32_t ioportmask_t**
Digital I/O port sized unsigned type.
- **typedef uint32_t iomode_t**
Digital I/O modes.
- **typedef uint32_t ioportid_t**
Port Identifier.

10.44.3 Function Documentation

10.44.3.1 `ioportmask_t palReadBus(IOBus *bus)`

Read from an I/O bus.

Note

The operation is not guaranteed to be atomic on all the architectures, for atomicity and/or portability reasons you may need to enclose port I/O operations between `chSysLock()` and `chSysUnlock()`.

The function internally uses the `palReadGroup()` macro. The use of this function is preferred when you value code size, readability and error checking over speed.

The function can be called from any context.

Parameters

in	<code>bus</code> the I/O bus, pointer to a <code>IOBus</code> structure
----	---

Returns

The bus logical states.

Function Class:

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

10.44.3.2 `void palWriteBus(IOBus *bus, ioportmask_t bits)`

Write to an I/O bus.

Note

The operation is not guaranteed to be atomic on all the architectures, for atomicity and/or portability reasons you may need to enclose port I/O operations between `chSysLock()` and `chSysUnlock()`.

The default implementation is non atomic and not necessarily optimal. Low level drivers may optimize the function by using specific hardware or coding.

The function can be called from any context.

Parameters

in	<code>bus</code> the I/O bus, pointer to a <code>IOBus</code> structure
in	<code>bits</code> the bits to be written on the I/O bus. Values exceeding the bus width are masked so most significant bits are lost.

Function Class:

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

10.44.3.3 `void palSetBusMode(IOBus *bus, iomode_t mode)`

Programs a bus with the specified mode.

Note

The operation is not guaranteed to be atomic on all the architectures, for atomicity and/or portability reasons you may need to enclose port I/O operations between `chSysLock()` and `chSysUnlock()`.

The default implementation is non atomic and not necessarily optimal. Low level drivers may optimize the function by using specific hardware or coding.

The function can be called from any context.

Parameters

in	<i>bus</i>	the I/O bus, pointer to a IOBus structure
in	<i>mode</i>	the mode

Function Class:

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

10.44.3.4 void _pal_lld_init (const PALConfig * config)

STM32 I/O ports configuration.

Ports A-D(E, F, G, H) clocks enabled.

Parameters

in	<i>config</i>	the STM32 ports configuration
----	---------------	-------------------------------

Function Class:

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

10.44.3.5 void _pal_lld_setgroupmode (ioportid_t port, ioportmask_t mask, iomode_t mode)

Pads mode setup.

This function programs a pads group belonging to the same port with the specified mode.

Parameters

in	<i>port</i>	the port identifier
in	<i>mask</i>	the group mask
in	<i>mode</i>	the mode

Function Class:

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

10.44.4 Define Documentation**10.44.4.1 #define PAL_MODE_RESET 0**

After reset state.

The state itself is not specified and is architecture dependent, it is guaranteed to be equal to the after-reset state. It is usually an input state.

10.44.4.2 #define PAL_MODE_UNCONNECTED 1

Safe state for **unconnected** pads.

The state itself is not specified and is architecture dependent, it may be mapped on **PAL_MODE_INPUT_PULLUP**, **PAL_MODE_INPUT_PULLDOWN** or **PAL_MODE_OUTPUT_PUSH_PULL** for example.

10.44.4.3 #define PAL_MODE_INPUT 2

Regular input high-Z pad.

10.44.4.4 #define PAL_MODE_INPUT_PULLUP 3

Input pad with weak pull up resistor.

10.44.4.5 #define PAL_MODE_INPUT_PULLDOWN 4

Input pad with weak pull down resistor.

10.44.4.6 #define PAL_MODE_INPUT_ANALOG 5

Analog input mode.

10.44.4.7 #define PAL_MODE_OUTPUT_PUSH_PULL 6

Push-pull output pad.

10.44.4.8 #define PAL_MODE_OUTPUT_OPENDRAIN 7

Open-drain output pad.

10.44.4.9 #define PAL_LOW 0

Logical low state.

10.44.4.10 #define PAL_HIGH 1

Logical high state.

10.44.4.11 #define PAL_PORT_BIT(*n*) ((ioportmask_t)(1 << (*n*)))

Port bit helper macro.

This macro calculates the mask of a bit within a port.

Parameters

in *n* bit position within the port

Returns

The bit mask.

10.44.4.12 #define PAL_GROUP_MASK(*width*) ((ioportmask_t)(1 << (*width*)) - 1)

Bits group mask helper.

This macro calculates the mask of a bits group.

Parameters

in *width* group width

Returns

The group mask.

```
10.44.4.13 #define _IOBUS_DATA( name, port, width, offset ) {port, PAL_GROUP_MASK(width), offset}
```

Data part of a static I/O bus initializer.

This macro should be used when statically initializing an I/O bus that is part of a bigger structure.

Parameters

in	<i>name</i>	name of the IOBus variable
in	<i>port</i>	I/O port descriptor
in	<i>width</i>	bus width in bits
in	<i>offset</i>	bus bit offset within the port

```
10.44.4.14 #define IOBUS_DECL( name, port, width, offset ) IOBus name = _IOBUS_DATA(name, port, width, offset)
```

Static I/O bus initializer.

Parameters

in	<i>name</i>	name of the IOBus variable
in	<i>port</i>	I/O port descriptor
in	<i>width</i>	bus width in bits
in	<i>offset</i>	bus bit offset within the port

```
10.44.4.15 #define palInit( config ) pal_lld_init(config)
```

PAL subsystem initialization.

Note

This function is implicitly invoked by [halInit\(\)](#), there is no need to explicitly initialize the driver.

Parameters

in	<i>config</i>	pointer to an architecture specific configuration structure. This structure is defined in the low level driver header.
----	---------------	--

Function Class:

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

```
10.44.4.16 #define palReadPort( port ) ((void)(port), 0)
```

Reads the physical I/O port states.

Note

The default implementation always return zero and computes the parameter eventual side effects.
The function can be called from any context.

Parameters

in	<i>port</i>	port identifier
----	-------------	-----------------

Returns

The port logical states.

Function Class:

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

```
10.44.4.17 #define palReadLatch( port ) ((void)(port), 0)
```

Reads the output latch.

The purpose of this function is to read back the latched output value.

Note

The default implementation always return zero and computes the parameter eventual side effects.
The function can be called from any context.

Parameters

in	port	port identifier
----	------	-----------------

Returns

The latched logical states.

Function Class:

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

```
10.44.4.18 #define palWritePort( port, bits ) ((void)(port), (void)(bits))
```

Writes a bits mask on a I/O port.

Note

The default implementation does nothing except computing the parameters eventual side effects.
The function can be called from any context.

Parameters

in	port	port identifier
in	bits	bits to be written on the specified port

Function Class:

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

```
10.44.4.19 #define palSetPort( port, bits ) palWritePort(port, palReadLatch(port) | (bits))
```

Sets a bits mask on a I/O port.

Note

The operation is not guaranteed to be atomic on all the architectures, for atomicity and/or portability reasons you may need to enclose port I/O operations between `chSysLock()` and `chSysUnlock()`.
The default implementation is non atomic and not necessarily optimal. Low level drivers may optimize the function by using specific hardware or coding.
The function can be called from any context.

Parameters

in	port	port identifier
in	bits	bits to be ORed on the specified port

Function Class:

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

```
10.44.4.20 #define palClearPort( port, bits ) palWritePort(port, palReadLatch(port) & ~(bits))
```

Clears a bits mask on a I/O port.

Note

The operation is not guaranteed to be atomic on all the architectures, for atomicity and/or portability reasons you may need to enclose port I/O operations between `chSysLock()` and `chSysUnlock()`.

The default implementation is non atomic and not necessarily optimal. Low level drivers may optimize the function by using specific hardware or coding.

The function can be called from any context.

Parameters

in	<i>port</i>	port identifier
in	<i>bits</i>	bits to be cleared on the specified port

Function Class:

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

```
10.44.4.21 #define palTogglePort( port, bits ) palWritePort(port, palReadLatch(port) ^ (bits))
```

Toggles a bits mask on a I/O port.

Note

The operation is not guaranteed to be atomic on all the architectures, for atomicity and/or portability reasons you may need to enclose port I/O operations between `chSysLock()` and `chSysUnlock()`.

The default implementation is non atomic and not necessarily optimal. Low level drivers may optimize the function by using specific hardware or coding.

The function can be called from any context.

Parameters

in	<i>port</i>	port identifier
in	<i>bits</i>	bits to be XORed on the specified port

Function Class:

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

```
10.44.4.22 #define palReadGroup( port, mask, offset ) ((palReadPort(port) >> (offset)) & (mask))
```

Reads a group of bits.

Note

The function can be called from any context.

Parameters

in	<i>port</i>	port identifier
in	<i>mask</i>	group mask, a logical AND is performed on the input data
in	<i>offset</i>	group bit offset within the port

Returns

The group logical states.

Function Class:

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

10.44.4.23 #define palWriteGroup(*port*, *mask*, *offset*, *bits*)

Value:

```
palWritePort(port, (palReadLatch(port) & ~((mask) << (offset))) | \
((bits) & (mask)) << (offset))) \
```

Writes a group of bits.

Note

The function can be called from any context.

Parameters

in	<i>port</i>	port identifier
in	<i>mask</i>	group mask, a logical AND is performed on the output data
in	<i>offset</i>	group bit offset within the port
in	<i>bits</i>	bits to be written. Values exceeding the group width are masked.

Function Class:

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

10.44.4.24 #define palSetGroupMode(*port*, *mask*, *offset*, *mode*)

Pads group mode setup.

This function programs a pads group belonging to the same port with the specified mode.

Note

Programming an unknown or unsupported mode is silently ignored.

The function can be called from any context.

Parameters

in	<i>port</i>	port identifier
in	<i>mask</i>	group mask
in	<i>offset</i>	group bit offset within the port
in	<i>mode</i>	group mode

Function Class:

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

10.44.4.25 #define palReadPad(*port*, *pad*) ((*palReadPort*(*port*) >> (*pad*)) & 1)

Reads an input pad logical state.

Note

The default implementation not necessarily optimal. Low level drivers may optimize the function by using specific hardware or coding.

The default implementation internally uses the [palReadPort \(\)](#).

The function can be called from any context.

Parameters

in	<i>port</i>	port identifier
in	<i>pad</i>	pad number within the port

Returns

The logical state.

Return values

PAL_LOW low logical state.

PAL_HIGH high logical state.

Function Class:

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

10.44.4.26 #define palWritePad(*port*, *pad*, *bit*)

Value:

```
palWritePort(port, (palReadLatch(port) & ~PAL_PORT_BIT(pad)) | \
    (((bit) & 1) << pad)) \
```

Writes a logical state on an output pad.

Note

The operation is not guaranteed to be atomic on all the architectures, for atomicity and/or portability reasons you may need to enclose port I/O operations between [chSysLock \(\)](#) and [chSysUnlock \(\)](#).

The default implementation is non atomic and not necessarily optimal. Low level drivers may optimize the function by using specific hardware or coding.

The default implementation internally uses the [palReadLatch \(\)](#) and [palWritePort \(\)](#).

The function can be called from any context.

Parameters

in	<i>port</i>	port identifier
in	<i>pad</i>	pad number within the port
in	<i>bit</i>	logical value, the value must be PAL_LOW or PAL_HIGH

Function Class:

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

10.44.4.27 #define palSetPad(*port*, *pad*) palSetPort(*port*, PAL_PORT_BIT(*pad*))

Sets a pad logical state to PAL_HIGH.

Note

The operation is not guaranteed to be atomic on all the architectures, for atomicity and/or portability reasons you may need to enclose port I/O operations between [chSysLock \(\)](#) and [chSysUnlock \(\)](#).

The default implementation is non atomic and not necessarily optimal. Low level drivers may optimize the function by using specific hardware or coding.

The default implementation internally uses the [palSetPort\(\)](#).

The function can be called from any context.

Parameters

in	<i>port</i>	port identifier
in	<i>pad</i>	pad number within the port

Function Class:

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

```
10.44.4.28 #define palClearPad( port, pad ) palClearPort(port, PAL_PORT_BIT(pad))
```

Clears a pad logical state to PAL_LOW.

Note

The operation is not guaranteed to be atomic on all the architectures, for atomicity and/or portability reasons you may need to enclose port I/O operations between [chSysLock\(\)](#) and [chSysUnlock\(\)](#).

The default implementation is non atomic and not necessarily optimal. Low level drivers may optimize the function by using specific hardware or coding.

The default implementation internally uses the [palClearPort\(\)](#).

The function can be called from any context.

Parameters

in	<i>port</i>	port identifier
in	<i>pad</i>	pad number within the port

Function Class:

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

```
10.44.4.29 #define palTogglePad( port, pad ) palTogglePort(port, PAL_PORT_BIT(pad))
```

Toggles a pad logical state.

Note

The operation is not guaranteed to be atomic on all the architectures, for atomicity and/or portability reasons you may need to enclose port I/O operations between [chSysLock\(\)](#) and [chSysUnlock\(\)](#).

The default implementation is non atomic and not necessarily optimal. Low level drivers may optimize the function by using specific hardware or coding.

The default implementation internally uses the [palTogglePort\(\)](#).

The function can be called from any context.

Parameters

in	<i>port</i>	port identifier
in	<i>pad</i>	pad number within the port

Function Class:

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

```
10.44.4.30 #define palSetPadMode( port, pad, mode ) palSetGroupMode(port, PAL_PORT_BIT(pad), 0, mode)
```

Pad mode setup.

This function programs a pad with the specified mode.

Note

The default implementation not necessarily optimal. Low level drivers may optimize the function by using specific hardware or coding.

Programming an unknown or unsupported mode is silently ignored.

The function can be called from any context.

Parameters

in	<i>port</i>	port identifier
in	<i>pad</i>	pad number within the port
in	<i>mode</i>	pad mode

Function Class:

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

```
10.44.4.31 #define PAL_IOPORTS_WIDTH 32
```

Width, in bits, of an I/O port.

```
10.44.4.32 #define PAL_WHOLE_PORT ((ioportmask_t)0xFFFFFFFF)
```

Whole port mask.

This macro specifies all the valid bits into a port.

```
10.44.4.33 #define IOPORT1 0
```

First I/O port identifier.

Low level drivers can define multiple ports, it is suggested to use this naming convention.

```
10.44.4.34 #define pal_lld_init( config ) _pal_lld_init(config)
```

Low level PAL subsystem initialization.

Parameters

in	<i>config</i>	architecture-dependent ports configuration
----	---------------	--

Function Class:

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

```
10.44.4.35 #define pal_lld_readport( port ) 0
```

Reads the physical I/O port states.

Parameters

in	<i>port</i>	port identifier
----	-------------	-----------------

Returns

The port bits.

Function Class:

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

10.44.4.36 #define pal_lld_readlatch(*port*) 0

Reads the output latch.

The purpose of this function is to read back the latched output value.

Parameters

in *port* port identifier

Returns

The latched logical states.

Function Class:

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

10.44.4.37 #define pal_lld_writeport(*port*, *bits*)

Writes a bits mask on a I/O port.

Parameters

in *port* port identifier

in *bits* bits to be written on the specified port

Function Class:

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

10.44.4.38 #define pal_lld_setport(*port*, *bits*)

Sets a bits mask on a I/O port.

Note

The [PAL Driver](#) provides a default software implementation of this functionality, implement this function if can optimize it by using special hardware functionalities or special coding.

Parameters

in *port* port identifier

in *bits* bits to be ORed on the specified port

Function Class:

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

```
10.44.4.39 #define pal_lld_clearport( port, bits )
```

Clears a bits mask on a I/O port.

Note

The [PAL Driver](#) provides a default software implementation of this functionality, implement this function if can optimize it by using special hardware functionalities or special coding.

Parameters

in	<i>port</i>	port identifier
in	<i>bits</i>	bits to be cleared on the specified port

Function Class:

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

```
10.44.4.40 #define pal_lld_toggleport( port, bits )
```

Toggles a bits mask on a I/O port.

Note

The [PAL Driver](#) provides a default software implementation of this functionality, implement this function if can optimize it by using special hardware functionalities or special coding.

Parameters

in	<i>port</i>	port identifier
in	<i>bits</i>	bits to be XORed on the specified port

Function Class:

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

```
10.44.4.41 #define pal_lld_readgroup( port, mask, offset ) 0
```

Reads a group of bits.

Note

The [PAL Driver](#) provides a default software implementation of this functionality, implement this function if can optimize it by using special hardware functionalities or special coding.

Parameters

in	<i>port</i>	port identifier
in	<i>mask</i>	group mask
in	<i>offset</i>	group bit offset within the port

Returns

The group logical states.

Function Class:

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

```
10.44.4.42 #define pal_lld_writegroup( port, mask, offset, bits ) (void)bits
```

Writes a group of bits.

Note

The [PAL Driver](#) provides a default software implementation of this functionality, implement this function if can optimize it by using special hardware functionalities or special coding.

Parameters

in	<i>port</i>	port identifier
in	<i>mask</i>	group mask
in	<i>offset</i>	group bit offset within the port
in	<i>bits</i>	bits to be written. Values exceeding the group width are masked.

Function Class:

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

```
10.44.4.43 #define pal_lld_setgroupmode( port, mask, offset, mode ) .pal_lld_setgroupmode(port, mask << offset, mode)
```

Pads group mode setup.

This function programs a pads group belonging to the same port with the specified mode.

Note

Programming an unknown or unsupported mode is silently ignored.

Parameters

in	<i>port</i>	port identifier
in	<i>mask</i>	group mask
in	<i>offset</i>	group bit offset within the port
in	<i>mode</i>	group mode

Function Class:

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

```
10.44.4.44 #define pal_lld_readpad( port, pad ) PAL_LOW
```

Reads a logical state from an I/O pad.

Note

The [PAL Driver](#) provides a default software implementation of this functionality, implement this function if can optimize it by using special hardware functionalities or special coding.

Parameters

in	<i>port</i>	port identifier
in	<i>pad</i>	pad number within the port

Returns

The logical state.

Return values

PAL_LOW low logical state.
PAL_HIGH high logical state.

Function Class:

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

10.44.4.45 #define pal_lld_writepad(*port*, *pad*, *bit*)

Writes a logical state on an output pad.

Note

This function is not meant to be invoked directly by the application code.

The [PAL Driver](#) provides a default software implementation of this functionality, implement this function if can optimize it by using special hardware functionalities or special coding.

Parameters

in	<i>port</i>	port identifier
in	<i>pad</i>	pad number within the port
in	<i>bit</i>	logical value, the value must be <i>PAL_LOW</i> or <i>PAL_HIGH</i>

Function Class:

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

10.44.4.46 #define pal_lld_setpad(*port*, *pad*)

Sets a pad logical state to *PAL_HIGH*.

Note

The [PAL Driver](#) provides a default software implementation of this functionality, implement this function if can optimize it by using special hardware functionalities or special coding.

Parameters

in	<i>port</i>	port identifier
in	<i>pad</i>	pad number within the port

Function Class:

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

10.44.4.47 #define pal_lld_clearpad(*port*, *pad*)

Clears a pad logical state to *PAL_LOW*.

Note

The [PAL Driver](#) provides a default software implementation of this functionality, implement this function if can optimize it by using special hardware functionalities or special coding.

Parameters

in	<i>port</i>	port identifier
in	<i>pad</i>	pad number within the port

Function Class:

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

10.44.4.48 #define pal_lld_togglepad(*port*, *pad*)

Toggles a pad logical state.

Note

The [PAL Driver](#) provides a default software implementation of this functionality, implement this function if can optimize it by using special hardware functionalities or special coding.

Parameters

in	<i>port</i>	port identifier
in	<i>pad</i>	pad number within the port

Function Class:

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

10.44.4.49 #define pal_lld_setpadmode(*port*, *pad*, *mode*)

Pad mode setup.

This function programs a pad with the specified mode.

Note

The [PAL Driver](#) provides a default software implementation of this functionality, implement this function if can optimize it by using special hardware functionalities or special coding.
Programming an unknown or unsupported mode is silently ignored.

Parameters

in	<i>port</i>	port identifier
in	<i>pad</i>	pad number within the port
in	<i>mode</i>	pad mode

Function Class:

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

10.44.5 Typedef Documentation

10.44.5.1 **typedef uint32_t ioportmask_t**

Digital I/O port sized unsigned type.

10.44.5.2 **typedef uint32_t iomode_t**

Digital I/O modes.

10.44.5.3 **typedef uint32_t ioportid_t**

Port Identifier.

This type can be a scalar or some kind of pointer, do not make any assumption about it, use the provided macros when populating variables of this type.

10.45 PWM Driver

10.45.1 Detailed Description

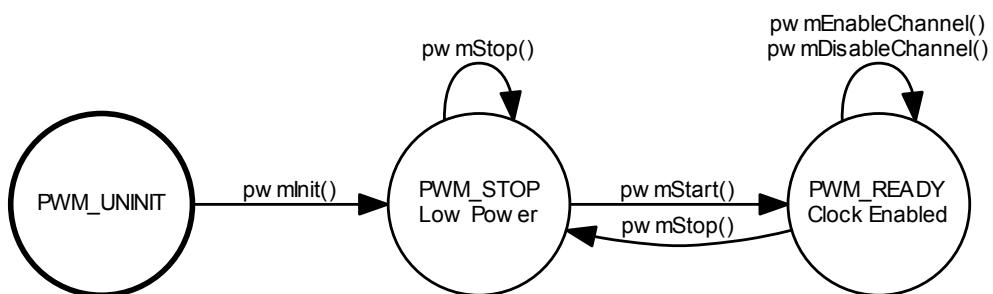
Generic PWM Driver. This module implements a generic PWM (Pulse Width Modulation) driver.

Precondition

In order to use the PWM driver the `HAL_USE_PWM` option must be enabled in `halconf.h`.

10.45.2 Driver State Machine

The driver implements a state machine internally, not all the driver functionalities can be used in any moment, any transition not explicitly shown in the following diagram has to be considered an error and shall be captured by an assertion (if enabled).



10.45.3 PWM Operations.

This driver abstracts a generic PWM timer composed of:

- A clock prescaler.
- A main up counter.
- A comparator register that resets the main counter to zero when the limit is reached. An optional callback can be generated when this happens.
- An array of `PWM_CHANNELS` PWM channels, each channel has an output, a comparator and is able to invoke an optional callback when a comparator match with the main counter happens.

A PWM channel output can be in two different states:

- **IDLE**, when the channel is disabled or after a match occurred.
- **ACTIVE**, when the channel is enabled and a match didn't occur yet in the current PWM cycle.

Note that the two states can be associated to both logical zero or one in the `PWMChannelConfig` structure.

Data Structures

- struct **PWMChannelConfig**
PWM driver channel configuration structure.
- struct **PWMConfig**
Driver configuration structure.
- struct **PWMDriver**
Structure representing an PWM driver.

Functions

- void **pwmInit** (void)
PWM Driver initialization.
- void **pwmObjectInit** (**PWMDriver** *pwmp)
*Initializes the standard part of a **PWMDriver** structure.*
- void **pwmStart** (**PWMDriver** *pwmp, const **PWMConfig** *config)
Configures and activates the PWM peripheral.
- void **pwmStop** (**PWMDriver** *pwmp)
Deactivates the PWM peripheral.
- void **pwmChangePeriod** (**PWMDriver** *pwmp, **pwmcnt_t** period)
Changes the period the PWM peripheral.
- void **pwmEnableChannel** (**PWMDriver** *pwmp, **pwmchannel_t** channel, **pwmcnt_t** width)
Enables a PWM channel.
- void **pwmDisableChannel** (**PWMDriver** *pwmp, **pwmchannel_t** channel)
Disables a PWM channel.
- void **pwm_lld_init** (void)
Low level PWM driver initialization.
- void **pwm_lld_start** (**PWMDriver** *pwmp)
Configures and activates the PWM peripheral.
- void **pwm_lld_stop** (**PWMDriver** *pwmp)
Deactivates the PWM peripheral.
- void **pwm_lld_change_period** (**PWMDriver** *pwmp, **pwmcnt_t** period)
Changes the period the PWM peripheral.
- void **pwm_lld_enable_channel** (**PWMDriver** *pwmp, **pwmchannel_t** channel, **pwmcnt_t** width)
Enables a PWM channel.
- void **pwm_lld_disable_channel** (**PWMDriver** *pwmp, **pwmchannel_t** channel)
Disables a PWM channel.

Variables

- **PWMDriver PWMD1**
PWM1 driver identifier.

PWM output mode macros

- #define **PWM_OUTPUT_MASK** 0x0F
Standard output modes mask.
- #define **PWM_OUTPUT_DISABLED** 0x00
Output not driven, callback only.
- #define **PWM_OUTPUT_ACTIVE_HIGH** 0x01
Positive PWM logic, active is logic level one.
- #define **PWM_OUTPUT_ACTIVE_LOW** 0x02
Inverse PWM logic, active is logic level zero.

PWM duty cycle conversion

- `#define PWM_FRACTION_TO_WIDTH(pwmp, denominator, numerator)`
Converts from fraction to pulse width.
- `#define PWM_DEGREES_TO_WIDTH(pwmp, degrees) PWM_FRACTION_TO_WIDTH(pwmp, 36000, degrees)`
Converts from degrees to pulse width.
- `#define PWM_PERCENTAGE_TO_WIDTH(pwmp, percentage) PWM_FRACTION_TO_WIDTH(pwmp, 10000, percentage)`
Converts from percentage to pulse width.

Macro Functions

- `#define pwmChangePeriodI(pwmp, value)`
Changes the period the PWM peripheral.
- `#define pwmEnableChannelI(pwmp, channel, width) pwm_lld_enable_channel(pwmp, channel, width)`
Enables a PWM channel.
- `#define pwmDisableChannelI(pwmp, channel) pwm_lld_disable_channel(pwmp, channel)`
Disables a PWM channel.
- `#define pwmIsChannelEnabledI(pwmp, channel) pwm_lld_is_channel_enabled(pwmp, channel)`
Returns a PWM channel status.

Configuration options

- `#define PLATFORM_PWM_USE_PWM1 FALSE`
PWM driver enable switch.

Defines

- `#define PWM_CHANNELS 4`
Number of PWM channels per PWM driver.
- `#define pwm_lld_is_channel_enabled(pwmp, channel) FALSE`
Returns a PWM channel status.

Typedefs

- `typedef struct PWMDriver PWMDriver`
Type of a structure representing a PWM driver.
- `typedef void(* pwcallback_t)(PWMDriver *pwmp)`
PWM notification callback type.
- `typedef uint32_t pwmmode_t`
PWM mode type.
- `typedef uint8_t pwmchannel_t`
PWM channel type.
- `typedef uint16_t pwmcnt_t`
PWM counter type.

Enumerations

- `enum pwmstate_t { PWM_UNINIT = 0, PWM_STOP = 1, PWM_READY = 2 }`
Driver state machine possible states.

10.45.4 Function Documentation

10.45.4.1 void pwmlInit(void)

PWM Driver initialization.

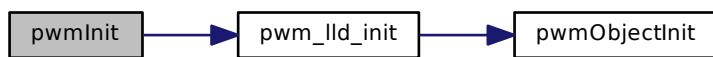
Note

This function is implicitly invoked by [halInit\(\)](#), there is no need to explicitly initialize the driver.

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:



10.45.4.2 void pwmObjectInit(PWMDriver * pwmp)

Initializes the standard part of a [PWMDriver](#) structure.

Parameters

out	<i>pwmp</i> pointer to a PWMDriver object
-----	---

Function Class:

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

10.45.4.3 void pwmStart(PWMDriver * pwmp, const PWMConfig * config)

Configures and activates the PWM peripheral.

Note

Starting a driver that is already in the `PWM_READY` state disables all the active channels.

Parameters

in	<i>pwmp</i> pointer to a PWMDriver object
in	<i>config</i> pointer to a PWMConfig 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:



10.45.4.4 void pwmStop (PWMDriver * pwmp)

Deactivates the PWM peripheral.

Parameters

in	<i>pwmp</i> pointer to a PWMDriver 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:



10.45.4.5 void pwmChangePeriod (PWMDriver * pwmp, pwcnt_t period)

Changes the period the PWM peripheral.

This function changes the period of a PWM unit that has already been activated using [pwmStart \(\)](#).

Precondition

The PWM unit must have been activated using [pwmStart \(\)](#).

Postcondition

The PWM unit period is changed to the new value.

Note

If a period is specified that is shorter than the pulse width programmed in one of the channels then the behavior is not guaranteed.

Parameters

in	<i>pwmp</i> pointer to a PWMDriver object
in	<i>period</i> new cycle time in ticks

Function Class:

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

10.45.4.6 void pwmEnableChannel (PWMDriver * *pwmp*, pwmchannel_t *channel*, pwcnt_t *width*)

Enables a PWM channel.

Precondition

The PWM unit must have been activated using [pwmStart \(\)](#).

Postcondition

The channel is active using the specified configuration.

Note

Depending on the hardware implementation this function has effect starting on the next cycle (recommended implementation) or immediately (fallback implementation).

Parameters

in	<i>pwmp</i>	pointer to a PWMDriver object
in	<i>channel</i>	PWM channel identifier (0...PWM_CHANNELS-1)
in	<i>width</i>	PWM pulse width as clock pulses number

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:

**10.45.4.7 void pwmDisableChannel (PWMDriver * *pwmp*, pwmchannel_t *channel*)**

Disables a PWM channel.

Precondition

The PWM unit must have been activated using [pwmStart \(\)](#).

Postcondition

The channel is disabled and its output line returned to the idle state.

Note

Depending on the hardware implementation this function has effect starting on the next cycle (recommended implementation) or immediately (fallback implementation).

Parameters

in	<i>pwmp</i> pointer to a PWMDriver object
in	<i>channel</i> PWM channel identifier (0...PWM_CHANNELS-1)

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:



10.45.4.8 void pwm_lld_init(void)

Low level PWM driver initialization.

Function Class:

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

Here is the call graph for this function:



10.45.4.9 void pwm_lld_start(PWMDriver * pwmp)

Configures and activates the PWM peripheral.

Parameters

in	<i>pwmp</i> pointer to the PWMDriver object
----	---

Function Class:

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

10.45.4.10 void pwm_lld_stop(PWMDriver * pwmp)

Deactivates the PWM peripheral.

Parameters

in *pwmp* pointer to the `PWMDriver` object

Function Class:

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

10.45.4.11 void pwm_lld_change_period (PWMDriver * *pwmp*, pwcnt_t *period*)

Changes the period the PWM peripheral.

This function changes the period of a PWM unit that has already been activated using `pwmStart()`.

Precondition

The PWM unit must have been activated using `pwmStart()`.

Postcondition

The PWM unit period is changed to the new value.

Note

The function has effect at the next cycle start.

If a period is specified that is shorter than the pulse width programmed in one of the channels then the behavior is not guaranteed.

Parameters

in *pwmp* pointer to a `PWMDriver` object
in *period* new cycle time in ticks

Function Class:

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

10.45.4.12 void pwm_lld_enable_channel (PWMDriver * *pwmp*, pwmchannel_t *channel*, pwcnt_t *width*)

Enables a PWM channel.

Precondition

The PWM unit must have been activated using `pwmStart()`.

Postcondition

The channel is active using the specified configuration.

Note

Depending on the hardware implementation this function has effect starting on the next cycle (recommended implementation) or immediately (fallback implementation).

Parameters

in *pwmp* pointer to a `PWMDriver` object
in *channel* PWM channel identifier (0...`PWM_CHANNELS-1`)
in *width* PWM pulse width as clock pulses number

Function Class:

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

10.45.4.13 void pwm_lld_disable_channel (**PWMDriver** * *pwmp*, **pwmchannel_t** *channel*)

Disables a PWM channel.

Precondition

The PWM unit must have been activated using [pwmStart \(\)](#).

Postcondition

The channel is disabled and its output line returned to the idle state.

Note

Depending on the hardware implementation this function has effect starting on the next cycle (recommended implementation) or immediately (fallback implementation).

Parameters

in	<i>pwmp</i>	pointer to a PWMDriver object
in	<i>channel</i>	PWM channel identifier (0...PWM_CHANNELS-1)

Function Class:

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

10.45.5 Variable Documentation

10.45.5.1 PWMDriver PWMD1

PWM1 driver identifier.

10.45.6 Define Documentation

10.45.6.1 #define PWM_OUTPUT_MASK 0x0F

Standard output modes mask.

10.45.6.2 #define PWM_OUTPUT_DISABLED 0x00

Output not driven, callback only.

10.45.6.3 #define PWM_OUTPUT_ACTIVE_HIGH 0x01

Positive PWM logic, active is logic level one.

10.45.6.4 #define PWM_OUTPUT_ACTIVE_LOW 0x02

Inverse PWM logic, active is logic level zero.

10.45.6.5 #define PWM_FRACTION_TO_WIDTH(*pwmp*, *denominator*, *numerator*)

Value:

```
((pwmcnt_t) (((pwmcnt_t)(pwmp)->period) *  
             (pwmcnt_t)(numerator)) / (pwmcnt_t)(denominator))) \
```

Converts from fraction to pulse width.

Note

Be careful with rounding errors, this is integer math not magic. You can specify tenths of thousandth but make sure you have the proper hardware resolution by carefully choosing the clock source and prescaler settings, see PWM_COMPUTE_PSC.

Parameters

in	<i>pwmp</i>	pointer to a PWMDriver object
in	<i>denominator</i>	denominator of the fraction
in	<i>numerator</i>	numerator of the fraction

Returns

The pulse width to be passed to [pwmEnableChannel\(\)](#).

Function Class:

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

10.45.6.6 #define PWM_DEGREES_TO_WIDTH(*pwmp*, *degrees*) PWM_FRACTION_TO_WIDTH(pwmp, 36000, degrees)

Converts from degrees to pulse width.

Note

Be careful with rounding errors, this is integer math not magic. You can specify hundredths of degrees but make sure you have the proper hardware resolution by carefully choosing the clock source and prescaler settings, see PWM_COMPUTE_PSC.

Parameters

in	<i>pwmp</i>	pointer to a PWMDriver object
in	<i>degrees</i>	degrees as an integer between 0 and 36000

Returns

The pulse width to be passed to [pwmEnableChannel\(\)](#).

Function Class:

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

10.45.6.7 #define PWM_PERCENTAGE_TO_WIDTH(*pwmp*, *percentage*) PWM_FRACTION_TO_WIDTH(pwmp, 10000, percentage)

Converts from percentage to pulse width.

Note

Be careful with rounding errors, this is integer math not magic. You can specify tenths of thousandth but make

sure you have the proper hardware resolution by carefully choosing the clock source and prescaler settings, see `PWM_COMPUTE_PSC`.

Parameters

in	<code>pwmp</code>	pointer to a <code>PWMDriver</code> object
in	<code>percentage</code>	percentage as an integer between 0 and 10000

Returns

The pulse width to be passed to `pwmEnableChannel()`.

Function Class:

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

10.45.6.8 #define `pwmChangePeriodI(pwmp, value)`

Value:

```
{
    (pwmp)->period = (value);
    pwm_lld_change_period(pwmp, value);
}
```

Changes the period the PWM peripheral.

This function changes the period of a PWM unit that has already been activated using `pwmStart()`.

Precondition

The PWM unit must have been activated using `pwmStart()`.

Postcondition

The PWM unit period is changed to the new value.

Note

If a period is specified that is shorter than the pulse width programmed in one of the channels then the behavior is not guaranteed.

Parameters

in	<code>pwmp</code>	pointer to a <code>PWMDriver</code> object
in	<code>value</code>	new cycle time in ticks

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.

10.45.6.9 #define `pwmEnableChannell(pwmp, channel, width) pwm_lld_enable_channel(pwmp, channel, width)`

Enables a PWM channel.

Precondition

The PWM unit must have been activated using `pwmStart()`.

Postcondition

The channel is active using the specified configuration.

Note

Depending on the hardware implementation this function has effect starting on the next cycle (recommended implementation) or immediately (fallback implementation).

Parameters

in	<i>pwmp</i>	pointer to a PWMDriver object
in	<i>channel</i>	PWM channel identifier (0...PWM_CHANNELS-1)
in	<i>width</i>	PWM pulse width as clock pulses number

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.

10.45.6.10 #define pwmDisableChannel(*pwmp*, *channel*) pwm_lld_disable_channel(*pwmp*, *channel*)

Disables a PWM channel.

Precondition

The PWM unit must have been activated using [pwmStart \(\)](#).

Postcondition

The channel is disabled and its output line returned to the idle state.

Note

Depending on the hardware implementation this function has effect starting on the next cycle (recommended implementation) or immediately (fallback implementation).

Parameters

in	<i>pwmp</i>	pointer to a PWMDriver object
in	<i>channel</i>	PWM channel identifier (0...PWM_CHANNELS-1)

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.

10.45.6.11 #define pwmlsChannelEnabled(*pwmp*, *channel*) pwm_lld_is_channel_enabled(*pwmp*, *channel*)

Returns a PWM channel status.

Precondition

The PWM unit must have been activated using [pwmStart \(\)](#).

Parameters

in	<i>pwmp</i>	pointer to a PWMDriver object
in	<i>channel</i>	PWM channel identifier (0...PWM_CHANNELS-1)

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.

10.45.6.12 `#define PWM_CHANNELS 4`

Number of PWM channels per PWM driver.

10.45.6.13 `#define PLATFORM_PWM_USE_PWM1 FALSE`

PWM driver enable switch.

If set to TRUE the support for PWM1 is included.

10.45.6.14 `#define pwm_lld_is_channel_enabled(pwmp, channel) FALSE`

Returns a PWM channel status.

Precondition

The PWM unit must have been activated using `pwmStart()`.

Parameters

in	<i>pwmp</i>	pointer to a <code>PWMDriver</code> object
in	<i>channel</i>	PWM channel identifier (0...PWM_CHANNELS-1)

Function Class:

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

10.45.7 Typedef Documentation

10.45.7.1 `typedef struct PWMDriver PWMDriver`

Type of a structure representing a PWM driver.

10.45.7.2 `typedef void(* pwmcallback_t)(PWMDriver *pwmp)`

PWM notification callback type.

Parameters

in	<i>pwmp</i>	pointer to a <code>PWMDriver</code> object
----	-------------	--

10.45.7.3 `typedef uint32_t pwmmode_t`

PWM mode type.

10.45.7.4 `typedef uint8_t pwmchannel_t`

PWM channel type.

10.45.7.5 `typedef uint16_t pwcnt_t`

PWM counter type.

10.45.8 Enumeration Type Documentation

10.45.8.1 enum pwmstate_t

Driver state machine possible states.

Enumerator:

PWM_UNINIT Not initialized.

PWM_STOP Stopped.

PWM_READY Ready.

10.46 RTC Driver

10.46.1 Detailed Description

Real Time Clock Abstraction Layer. This module defines an abstract interface for a Real Time Clock Peripheral.

Precondition

In order to use the RTC driver the `HAL_USE_RTC` option must be enabled in `halconf.h`.

Functions

- void `rtcInit` (void)
RTC Driver initialization.
- void `rtcSetTime` (RTCDriver *rtcp, const RTCTime *timespec)
Set current time.
- void `rtcGetTime` (RTCDriver *rtcp, RTCTime *timespec)
Get current time.
- uint32_t `rtcGetTimeFat` (RTCDriver *rtcp)
Get current time in format suitable for usage in FatFS.
- void `rtcSetAlarm` (RTCDriver *rtcp, rtcalarm_t alarm, const RTCAlarm *alarmspec)
Set alarm time.
- void `rtcGetAlarm` (RTCDriver *rtcp, rtcalarm_t alarm, RTCAlarm *alarmspec)
Get current alarm.
- void `rtcSetCallback` (RTCDriver *rtcp, rtccb_t callback)
Enables or disables RTC callbacks.

Date/Time bit masks

- #define `RTC_TIME_SECONDS_MASK` 0x0000001F
- #define `RTC_TIME_MINUTES_MASK` 0x000007E0
- #define `RTC_TIME_HOURS_MASK` 0x0000F800
- #define `RTC_DATE_DAYS_MASK` 0x001F0000
- #define `RTC_DATE_MONTHS_MASK` 0x01E00000
- #define `RTC_DATE_YEARS_MASK` 0xFE000000

Defines

- `#define rtcSetTimel(rtcp, timespec) rtc_lld_set_time(rtcp, timespec)`
Set current time.
- `#define rtcGetTimel(rtcp, timespec) rtc_lld_get_time(rtcp, timespec)`
Get current time.
- `#define rtcSetAlarml(rtcp, alarm, alarmspec) rtc_lld_set_alarm(rtcp, alarm, alarmspec)`
Set alarm time.
- `#define rtcGetAlarml(rtcp, alarm, alarmspec) rtc_lld_get_alarm(rtcp, alarm, alarmspec)`
Get current alarm.
- `#define rtcSetCallbackl(rtcp, callback) rtc_lld_set_callback(rtcp, callback)`
Enables or disables RTC callbacks.

Typedefs

- `typedef struct RTCDriver RTCDriver`
Type of a structure representing an RTC driver.
- `typedef struct RTCTime RTCTime`
Type of a structure representing an RTC time stamp.

10.46.2 Function Documentation

10.46.2.1 void rtcInit (void)

RTC Driver initialization.

Note

This function is implicitly invoked by `halInit ()`, there is no need to explicitly initialize the driver.

Function Class:

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

10.46.2.2 void rtcSetTime (RTCDriver * rtcp, const RTCTime * timespec)

Set current time.

Parameters

in	<code>rtcp</code>	pointer to RTC driver structure
in	<code>timespec</code>	pointer to a RTCTime structure

Function Class:

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

10.46.2.3 void rtcGetTime (RTCDriver * rtcp, RTCTime * timespec)

Get current time.

Parameters

in	<code>rtcp</code>	pointer to RTC driver structure
out	<code>timespec</code>	pointer to a RTCTime structure

Function Class:

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

10.46.2.4 uint32_t rtcGetTimeFat (RTCDriver * *rtcp*)

Get current time in format suitable for usage in FatFS.

Parameters

in *rtcp* pointer to RTC driver structure

Returns

FAT time value.

Function Class:

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

10.46.2.5 void rtcSetAlarm (RTCDriver * *rtcp*, rtcalarm_t *alarm*, const RTCAlarm * *alarmspec*)

Set alarm time.

Parameters

in *rtcp* pointer to RTC driver structure

in *alarm* alarm identifier

in *alarmspec* pointer to a RTCAlarm structure or NULL

Function Class:

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

10.46.2.6 void rtcGetAlarm (RTCDriver * *rtcp*, rtcalarm_t *alarm*, RTCAlarm * *alarmspec*)

Get current alarm.

Note

If an alarm has not been set then the returned alarm specification is not meaningful.

Parameters

in *rtcp* pointer to RTC driver structure

in *alarm* alarm identifier

out *alarmspec* pointer to a RTCAlarm structure

Function Class:

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

10.46.2.7 void rtcSetCallback (RTCDriver * *rtcp*, rtccb_t *callback*)

Enables or disables RTC callbacks.

This function enables or disables the callback, use a NULL pointer in order to disable it.

Parameters

in	<i>rtcp</i>	pointer to RTC driver structure
in	<i>callback</i>	callback function pointer or NULL

Function Class:

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

10.46.3 Define Documentation

10.46.3.1 #define rtcSetTime(*rtcp*, *timespec*) rtc_lld_set_time(*rtcp*, *timespec*)

Set current time.

Parameters

in	<i>rtcp</i>	pointer to RTC driver structure
in	<i>timespec</i>	pointer to a RTCTime 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.

10.46.3.2 #define rtcGetTime(*rtcp*, *timespec*) rtc_lld_get_time(*rtcp*, *timespec*)

Get current time.

Parameters

in	<i>rtcp</i>	pointer to RTC driver structure
out	<i>timespec</i>	pointer to a RTCTime 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.

10.46.3.3 #define rtcSetAlarm(*rtcp*, *alarm*, *alarmspec*) rtc_lld_set_alarm(*rtcp*, *alarm*, *alarmspec*)

Set alarm time.

Parameters

in	<i>rtcp</i>	pointer to RTC driver structure
in	<i>alarm</i>	alarm identifier
in	<i>alarmspec</i>	pointer to a RTCAlarm structure or NULL

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.

10.46.3.4 #define rtcGetAlarm(*rtcp*, *alarm*, *alarmspec*) rtc_lld_get_alarm(*rtcp*, *alarm*, *alarmspec*)

Get current alarm.

Note

If an alarm has not been set then the returned alarm specification is not meaningful.

Parameters

in	<i>rtcp</i>	pointer to RTC driver structure
in	<i>alarm</i>	alarm identifier
out	<i>alarmspec</i>	pointer to a RTCAlarm 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.

10.46.3.5 #define rtcSetCallback(*rtcp*, *callback*) *rtc_lld.set_callback(rtcp, callback)*

Enables or disables RTC callbacks.

This function enables or disables the callback, use a `NULL` pointer in order to disable it.

Parameters

in	<i>rtcp</i>	pointer to RTC driver structure
in	<i>callback</i>	callback function pointer or <code>NULL</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.

10.46.4 Typedef Documentation**10.46.4.1 typedef struct RTCDriver RTCDriver**

Type of a structure representing an RTC driver.

10.46.4.2 typedef struct RTCTime RTCTime

Type of a structure representing an RTC time stamp.

10.47 SDC Driver

10.47.1 Detailed Description

Generic SD Card Driver. This module implements a generic SDC (Secure Digital Card) driver.

Precondition

In order to use the SDC driver the `HAL_USE_SDC` option must be enabled in `halconf.h`.

10.47.2 Driver State Machine

This driver implements a state machine internally, see the [Abstract I/O Block Device](#) module documentation for details.

10.47.3 Driver Operations

This driver allows to read or write single or multiple 512 bytes blocks on a SD Card.

Data Structures

- struct [SDCConfig](#)
Driver configuration structure.
- struct [SDCDriverVMT](#)
SDCDriver virtual methods table.
- struct [SDCDriver](#)
Structure representing an SDC driver.

Functions

- void [sdclInit](#) (void)
SDC Driver initialization.
- void [sdcObjectInit](#) ([SDCDriver](#) *sdcp)
Initializes the standard part of a [SDCDriver](#) structure.
- void [sdcStart](#) ([SDCDriver](#) *sdcp, const [SDCConfig](#) *config)
Configures and activates the SDC peripheral.
- void [sdcStop](#) ([SDCDriver](#) *sdcp)
Deactivates the SDC peripheral.
- [bool_t](#) [sdcConnect](#) ([SDCDriver](#) *sdcp)
Performs the initialization procedure on the inserted card.
- [bool_t](#) [sdcDisconnect](#) ([SDCDriver](#) *sdcp)
Brings the driver in a state safe for card removal.
- [bool_t](#) [sdcRead](#) ([SDCDriver](#) *sdcp, uint32_t startblk, uint8_t *buf, uint32_t n)
Reads one or more blocks.
- [bool_t](#) [sdcWrite](#) ([SDCDriver](#) *sdcp, uint32_t startblk, const uint8_t *buf, uint32_t n)
Writes one or more blocks.
- [sdcflags_t](#) [sdcGetAndClearErrors](#) ([SDCDriver](#) *sdcp)
Returns the errors mask associated to the previous operation.
- [bool_t](#) [sdcSync](#) ([SDCDriver](#) *sdcp)
Waits for card idle condition.
- [bool_t](#) [sdcGetInfo](#) ([SDCDriver](#) *sdcp, [BlockDeviceInfo](#) *bdip)
Returns the media info.
- [bool_t](#) [sdcErase](#) ([SDCDriver](#) *sdcp, uint32_t startblk, uint32_t endblk)
Erases the supplied blocks.
- [bool_t](#) [_sdc_wait_for_transfer_state](#) ([SDCDriver](#) *sdcp)
Wait for the card to complete pending operations.
- void [sdc_lld_init](#) (void)
Low level SDC driver initialization.
- void [sdc_lld_start](#) ([SDCDriver](#) *sdcp)
Configures and activates the SDC peripheral.
- void [sdc_lld_stop](#) ([SDCDriver](#) *sdcp)
Deactivates the SDC peripheral.
- void [sdc_lld_start_clk](#) ([SDCDriver](#) *sdcp)
Starts the SDIO clock and sets it to init mode (400kHz or less).
- void [sdc_lld_set_data_clk](#) ([SDCDriver](#) *sdcp)
Sets the SDIO clock to data mode (25MHz or less).

- void `sdc_lld_stop_clk (SDCDriver *sdcp)`
Stops the SDIO clock.
- void `sdc_lld_set_bus_mode (SDCDriver *sdcp, sdcbusmode_t mode)`
Switches the bus to 4 bits mode.
- void `sdc_lld_send_cmd_none (SDCDriver *sdcp, uint8_t cmd, uint32_t arg)`
Sends an SDIO command with no response expected.
- `bool_t sdc_lld_send_cmd_short (SDCDriver *sdcp, uint8_t cmd, uint32_t arg, uint32_t *resp)`
Sends an SDIO command with a short response expected.
- `bool_t sdc_lld_send_cmd_short_crc (SDCDriver *sdcp, uint8_t cmd, uint32_t arg, uint32_t *resp)`
Sends an SDIO command with a short response expected and CRC.
- `bool_t sdc_lld_send_cmd_long_crc (SDCDriver *sdcp, uint8_t cmd, uint32_t arg, uint32_t *resp)`
Sends an SDIO command with a long response expected and CRC.
- `bool_t sdc_lld_read (SDCDriver *sdcp, uint32_t startblk, uint8_t *buf, uint32_t n)`
Reads one or more blocks.
- `bool_t sdc_lld_write (SDCDriver *sdcp, uint32_t startblk, const uint8_t *buf, uint32_t n)`
Writes one or more blocks.
- `bool_t sdc_lld_sync (SDCDriver *sdcp)`
Waits for card idle condition.

Variables

- `SDCDriver SDCD1`
SDCD1 driver identifier.

SD cart types

- `#define SDC_MODE_CARDTYPE_MASK 0xF`
Card type mask.
- `#define SDC_MODE_CARDTYPE_SDV11 0`
Card is SD V1.1.
- `#define SDC_MODE_CARDTYPE_SDV20 1`
Card is SD V2.0.
- `#define SDC_MODE_CARDTYPE_MMC 2`
Card is MMC.
- `#define SDC_MODE_HIGH_CAPACITY 0x10`
High cap.card.

SDC bus error conditions

- `#define SDC_NO_ERROR 0`
No error.
- `#define SDC_CMD_CRC_ERROR 1`
Command CRC error.
- `#define SDC_DATA_CRC_ERROR 2`
Data CRC error.
- `#define SDC_DATA_TIMEOUT 4`
HW write timeout.
- `#define SDC_COMMAND_TIMEOUT 8`
HW read timeout.
- `#define SDC_TX_UNDERRUN 16`

- `#define SDC_RX_OVERRUN 32`
RX buffer overrun.
- `#define SDC_STARTBIT_ERROR 64`
Start bit missing.
- `#define SDC_OVERFLOW_ERROR 128`
Card overflow error.
- `#define SDC_UNHANDLED_ERROR 0xFFFFFFFF`

SDC configuration options

- `#define SDC_INIT_RETRY 100`
Number of initialization attempts before rejecting the card.
- `#define SDC_MMC_SUPPORT FALSE`
Include support for MMC cards.
- `#define SDC_NICE_WAITING TRUE`
Delays insertions.

Macro Functions

- `#define sdclsCardInserted(sdcp) (sdc_lld_is_card_inserted(sdcp))`
Returns the card insertion status.
- `#define sdclsWriteProtected(sdcp) (sdc_lld_is_write_protected(sdcp))`
Returns the write protect status.

Configuration options

- `#define PLATFORM_SDC_USE_SDC1 TRUE`
SDC driver enable switch.

R1 response utilities

- `#define MMCSD_R1_ERROR(r1) (((r1) & MMCSD_R1_ERROR_MASK) != 0)`
Evaluates to TRUE if the R1 response contains error flags.
- `#define MMCSD_R1_STS(r1) (((r1) >> 9) & 15)`
Returns the status field of an R1 response.
- `#define MMCSD_R1_IS_CARD_LOCKED(r1) (((r1) >> 21) & 1)`
Evaluates to TRUE if the R1 response indicates a locked card.

Defines

- `#define _sdc_driver_methods _mmcisd_block_device_methods`
SDCDriver specific methods.

Typedefs

- `typedef uint32_t sdcmode_t`
Type of card flags.
- `typedef uint32_t sdctflags_t`
SDC Driver condition flags type.
- `typedef struct SDCDriver SDCDriver`
Type of a structure representing an SDC driver.

Enumerations

- enum `sdcbusmode_t`
Type of SDIO bus mode.

10.47.4 Function Documentation

10.47.4.1 void sdcInit(void)

SDC Driver initialization.

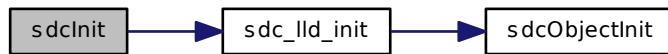
Note

This function is implicitly invoked by `halInit()`, there is no need to explicitly initialize the driver.

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:



10.47.4.2 void sdcObjectInit(SDCDriver * sdc)

Initializes the standard part of a `SDCDriver` structure.

Parameters

out	<code>sdc</code> pointer to the <code>SDCDriver</code> object
-----	---

Function Class:

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

10.47.4.3 void sdcStart(SDCDriver * sdc, const SDCCConfig * config)

Configures and activates the SDC peripheral.

Parameters

in	<code>sdc</code> pointer to the <code>SDCDriver</code> object
in	<code>config</code> pointer to the <code>SDCCConfig</code> object, can be <code>NULL</code> if the driver supports a default configuration or requires no configuration

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:



10.47.4.4 void sdcStop (SDCDriver * sdc)

Deactivates the SDC peripheral.

Parameters

in *sdc* pointer to the [SDCDriver](#) 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:



10.47.4.5 bool_t sdcConnect (SDCDriver * sdc)

Performs the initialization procedure on the inserted card.

This function should be invoked when a card is inserted and brings the driver in the `BLK_READY` state where it is possible to perform read and write operations.

Parameters

in *sdc* pointer to the [SDCDriver](#) object

Returns

The operation status.

Return values

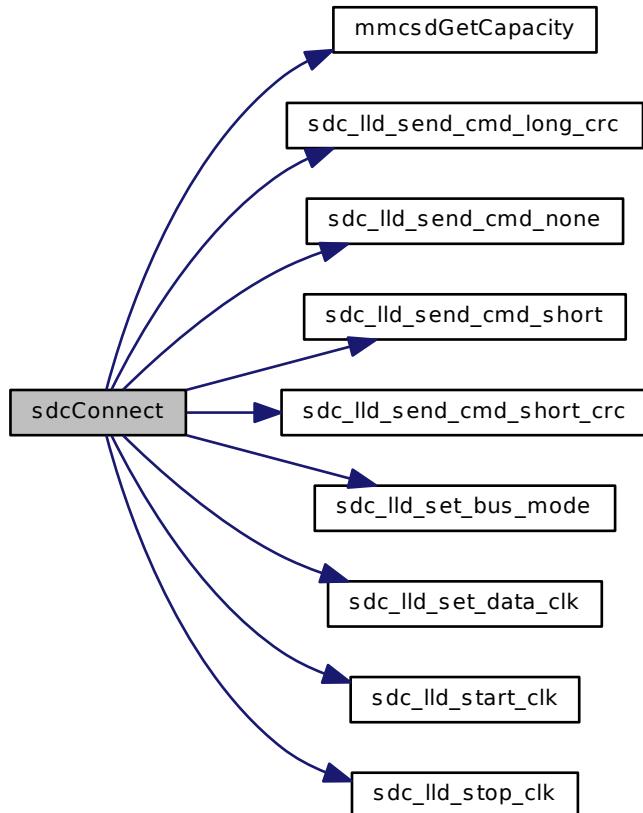
`CH_SUCCESS` operation succeeded.

`CH_FAILED` operation 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:



10.47.4.6 `bool_t sdcDisconnect(SDCDriver * sdc)`

Brings the driver in a state safe for card removal.

Parameters

`in` `sdc` pointer to the `SDCDriver` object

Returns

The operation status.

Return values

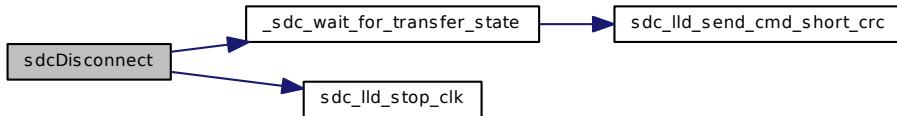
`CH_SUCCESS` operation succeeded.

`CH_FAILED` operation 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:



10.47.4.7 `bool_t sdcRead(SDCDriver *sdcp, uint32_t startblk, uint8_t *buf, uint32_t n)`

Reads one or more blocks.

Precondition

The driver must be in the `BLK_READY` state after a successful `sdcConnect()` invocation.

Parameters

in	<code>sdcp</code>	pointer to the <code>SDCDriver</code> object
in	<code>startblk</code>	first block to read
out	<code>buf</code>	pointer to the read buffer
in	<code>n</code>	number of blocks to read

Returns

The operation status.

Return values

<code>CH_SUCCESS</code>	operation succeeded.
<code>CH_FAILED</code>	operation 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:



10.47.4.8 `bool_t sdcWrite(SDCDriver *sdcp, uint32_t startblk, const uint8_t *buf, uint32_t n)`

Writes one or more blocks.

Precondition

The driver must be in the `BLK_READY` state after a successful `sdcConnect()` invocation.

Parameters

in	<code>sdcp</code>	pointer to the <code>SDCDriver</code> object
in	<code>startblk</code>	first block to write
out	<code>buf</code>	pointer to the write buffer
in	<code>n</code>	number of blocks to write

Returns

The operation status.

Return values

<code>CH_SUCCESS</code>	operation succeeded.
<code>CH_FAILED</code>	operation 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:



10.47.4.9 `sdcflags_t sdcGetAndClearErrors(SDCDriver *sdcp)`

Returns the errors mask associated to the previous operation.

Parameters

in	<code>sdcp</code>	pointer to the <code>SDCDriver</code> object
----	-------------------	--

Returns

The errors mask.

Function Class:

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

10.47.4.10 `bool_t sdcSync(SDCDriver *sdcp)`

Waits for card idle condition.

Parameters

in *sdcp* pointer to the `SDCDriver` object

Returns

The operation status.

Return values

CH_SUCCESS the operation succeeded.

CH_FAILED the operation 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:

**10.47.4.11 bool_t sdcGetInfo (SDCDriver * sdc, BlockDeviceInfo * bdip)**

Returns the media info.

Parameters

in *sdc* pointer to the `SDCDriver` object

out *bdip* pointer to a `BlockDeviceInfo` structure

Returns

The operation status.

Return values

CH_SUCCESS the operation succeeded.

CH_FAILED the operation failed.

Function Class:

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

10.47.4.12 bool_t sdcErase (SDCDriver * sdc, uint32_t startblk, uint32_t endblk)

Erases the supplied blocks.

Parameters

in *sdc* pointer to the `SDCDriver` object

in *startblk* starting block number

in *endblk* ending block number

Returns

The operation status.

Return values

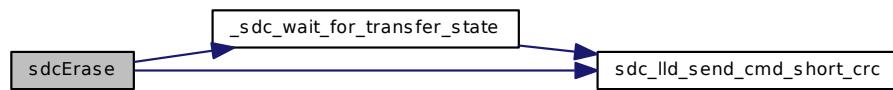
CH_SUCCESS the operation succeeded.

CH_FAILED the operation 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:

**10.47.4.13 bool_t _sdc_wait_for_transfer_state (SDCDriver * sdc)**

Wait for the card to complete pending operations.

Parameters

in *sdc* pointer to the `SDCDriver` object

Returns

The operation status.

Return values

CH_SUCCESS operation succeeded.

CH_FAILED operation failed.

Function Class:

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

Here is the call graph for this function:



10.47.4.14 void sdc_lld_init(void)

Low level SDC driver initialization.

Function Class:

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

Here is the call graph for this function:

**10.47.4.15 void sdc_lld_start(SDCDriver * sdc)**

Configures and activates the SDC peripheral.

Parameters

in *sdc* pointer to the [SDCDriver](#) object

Function Class:

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

10.47.4.16 void sdc_lld_stop(SDCDriver * sdc)

Deactivates the SDC peripheral.

Parameters

in *sdc* pointer to the [SDCDriver](#) object

Function Class:

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

10.47.4.17 void sdc_lld_start_clk(SDCDriver * sdc)

Starts the SDIO clock and sets it to init mode (400kHz or less).

Parameters

in *sdc* pointer to the [SDCDriver](#) object

Function Class:

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

10.47.4.18 void sdc_lld_set_data_clk (SDCDriver * sdc)

Sets the SDIO clock to data mode (25MHz or less).

Parameters

in *sdc* pointer to the [SDCDriver](#) object

Function Class:

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

10.47.4.19 void sdc_lld_stop_clk (SDCDriver * sdc)

Stops the SDIO clock.

Parameters

in *sdc* pointer to the [SDCDriver](#) object

Function Class:

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

10.47.4.20 void sdc_lld_set_bus_mode (SDCDriver * sdc, sdcbusmode_t mode)

Switches the bus to 4 bits mode.

Parameters

in *sdc* pointer to the [SDCDriver](#) object
in *mode* bus mode

Function Class:

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

10.47.4.21 void sdc_lld_send_cmd_none (SDCDriver * sdc, uint8_t cmd, uint32_t arg)

Sends an SDIO command with no response expected.

Parameters

in *sdc* pointer to the [SDCDriver](#) object
in *cmd* card command
in *arg* command argument

Function Class:

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

10.47.4.22 bool_t sdc_lld_send_cmd_short (SDCDriver * sdc, uint8_t cmd, uint32_t arg, uint32_t * resp)

Sends an SDIO command with a short response expected.

Note

The CRC is not verified.

Parameters

in	<i>sdcp</i>	pointer to the <code>SDCDriver</code> object
in	<i>cmd</i>	card command
in	<i>arg</i>	command argument
out	<i>resp</i>	pointer to the response buffer (one word)

Returns

The operation status.

Return values

<i>CH_SUCCESS</i>	operation succeeded.
<i>CH FAILED</i>	operation failed.

Function Class:

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

10.47.4.23 `bool_t sdc_lld_send_cmd_short_crc (SDCDriver * sdc, uint8_t cmd, uint32_t arg, uint32_t * resp)`

Sends an SDIO command with a short response expected and CRC.

Parameters

in	<i>sdc</i>	pointer to the <code>SDCDriver</code> object
in	<i>cmd</i>	card command
in	<i>arg</i>	command argument
out	<i>resp</i>	pointer to the response buffer (one word)

Returns

The operation status.

Return values

<i>CH_SUCCESS</i>	operation succeeded.
<i>CH FAILED</i>	operation failed.

Function Class:

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

10.47.4.24 `bool_t sdc_lld_send_cmd_long_crc (SDCDriver * sdc, uint8_t cmd, uint32_t arg, uint32_t * resp)`

Sends an SDIO command with a long response expected and CRC.

Parameters

in	<i>sdc</i>	pointer to the <code>SDCDriver</code> object
in	<i>cmd</i>	card command
in	<i>arg</i>	command argument
out	<i>resp</i>	pointer to the response buffer (four words)

Returns

The operation status.

Return values

<i>CH_SUCCESS</i>	operation succeeded.
<i>CH FAILED</i>	operation failed.

Function Class:

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

10.47.4.25 bool_t sdc_lld_read (SDCDriver * sdcp, uint32_t startblk, uint8_t * buf, uint32_t n)

Reads one or more blocks.

Parameters

in	<i>sdcp</i>	pointer to the SDCDriver object
in	<i>startblk</i>	first block to read
out	<i>buf</i>	pointer to the read buffer
in	<i>n</i>	number of blocks to read

Returns

The operation status.

Return values

<i>CH_SUCCESS</i>	operation succeeded.
<i>CH_FAILED</i>	operation failed.

Function Class:

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

10.47.4.26 bool_t sdc_lld_write (SDCDriver * sdcp, uint32_t startblk, const uint8_t * buf, uint32_t n)

Writes one or more blocks.

Parameters

in	<i>sdcp</i>	pointer to the SDCDriver object
in	<i>startblk</i>	first block to write
out	<i>buf</i>	pointer to the write buffer
in	<i>n</i>	number of blocks to write

Returns

The operation status.

Return values

<i>CH_SUCCESS</i>	operation succeeded.
<i>CH_FAILED</i>	operation failed.

Function Class:

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

10.47.4.27 bool_t sdc_lld_sync (SDCDriver * sdcp)

Waits for card idle condition.

Parameters

in	<i>sdcp</i>	pointer to the SDCDriver object
----	-------------	---

Returns

The operation status.

Return values

CH_SUCCESS the operation succeeded.
CH_FAILED the operation failed.

Function Class:

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

10.47.5 Variable Documentation

10.47.5.1 SDCDriver SD_CD1

SD_CD1 driver identifier.

10.47.6 Define Documentation

10.47.6.1 #define SDC_MODE_CARDTYPE_MASK 0xF

Card type mask.

10.47.6.2 #define SDC_MODE_CARDTYPE_SDV11 0

Card is SD V1.1.

10.47.6.3 #define SDC_MODE_CARDTYPE_SDV20 1

Card is SD V2.0.

10.47.6.4 #define SDC_MODE_CARDTYPE_MMC 2

Card is MMC.

10.47.6.5 #define SDC_MODE_HIGH_CAPACITY 0x10

High cap.card.

10.47.6.6 #define SDC_NO_ERROR 0

No error.

10.47.6.7 #define SDC_CMD_CRC_ERROR 1

Command CRC error.

10.47.6.8 #define SDC_DATA_CRC_ERROR 2

Data CRC error.

10.47.6.9 #define SDC_DATA_TIMEOUT 4

HW write timeout.

10.47.6.10 #define SDC_COMMAND_TIMEOUT 8

HW read timeout.

10.47.6.11 #define SDC_TX_UNDERRUN 16

TX buffer underrun.

10.47.6.12 #define SDC_RX_OVERRUN 32

RX buffer overrun.

10.47.6.13 #define SDC_STARTBIT_ERROR 64

Start bit missing.

10.47.6.14 #define SDC_OVERFLOW_ERROR 128

Card overflow error.

10.47.6.15 #define SDC_INIT_RETRY 100

Number of initialization attempts before rejecting the card.

Note

Attempts are performed at 10mS intervals.

10.47.6.16 #define SDC_MMCC_SUPPORT FALSE

Include support for MMC cards.

Note

MMC support is not yet implemented so this option must be kept at FALSE.

10.47.6.17 #define SDC_NICE_WAITING TRUE

Delays insertions.

If enabled this options inserts delays into the MMC waiting routines releasing some extra CPU time for the threads with lower priority, this may slow down the driver a bit however.

10.47.6.18 #define sdclsCardInserted(*sdcp*) (sdc_lld_is_card_inserted(*sdcp*))

Returns the card insertion status.

Note

This macro wraps a low level function named `sdc_lld_is_card_inserted()`, this function must be provided by the application because it is not part of the SDC driver.

Parameters

in *sdcp* pointer to the `SDCDriver` object

Returns

The card state.

Return values

FALSE card not inserted.
TRUE card inserted.

Function Class:

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

10.47.6.19 #define sdclsWriteProtected(*sdcp*) (sdc_lld_is_write_protected(*sdcp*))

Returns the write protect status.

Note

This macro wraps a low level function named `sdc_lld_is_write_protected()`, this function must be provided by the application because it is not part of the SDC driver.

Parameters

in *sdcp* pointer to the `SDCDriver` object

Returns

The card state.

Return values

FALSE not write protected.
TRUE write protected.

Function Class:

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

10.47.6.20 #define PLATFORM_SDC_USE_SDC1 TRUE

SDC driver enable switch.

If set to `TRUE` the support for SDC1 is included.

10.47.6.21 #define _sdc_driver_methods _mmcisd_block_device_methods

`SDCDriver` specific methods.

```
10.47.6.22 #define MMCSD_R1_ERROR( r1 ) (((r1) & MMCSD_R1_ERROR_MASK) != 0)
```

Evaluates to TRUE if the R1 response contains error flags.

Parameters

in *r1* the r1 response

```
10.47.6.23 #define MMCSD_R1_STS( r1 ) (((r1) >> 9) & 15)
```

Returns the status field of an R1 response.

Parameters

in *r1* the r1 response

```
10.47.6.24 #define MMCSD_R1_IS_CARD_LOCKED( r1 ) (((r1) >> 21) & 1)
```

Evaluates to TRUE if the R1 response indicates a locked card.

Parameters

in *r1* the r1 response

10.47.7 Typedef Documentation

```
10.47.7.1 typedef uint32_t sdcmode_t
```

Type of card flags.

```
10.47.7.2 typedef uint32_t sdcflags_t
```

SDC Driver condition flags type.

```
10.47.7.3 typedef struct SDCDriver SDCDriver
```

Type of a structure representing an SDC driver.

10.47.8 Enumeration Type Documentation

```
10.47.8.1 enum sdcbusmode_t
```

Type of SDIO bus mode.

10.48 Serial Driver

10.48.1 Detailed Description

Generic Serial Driver. This module implements a generic full duplex serial driver. The driver implements a [SerialDriver](#) interface and uses I/O Queues for communication between the upper and the lower driver. Event flags are used to notify the application about incoming data, outgoing data and other I/O events.

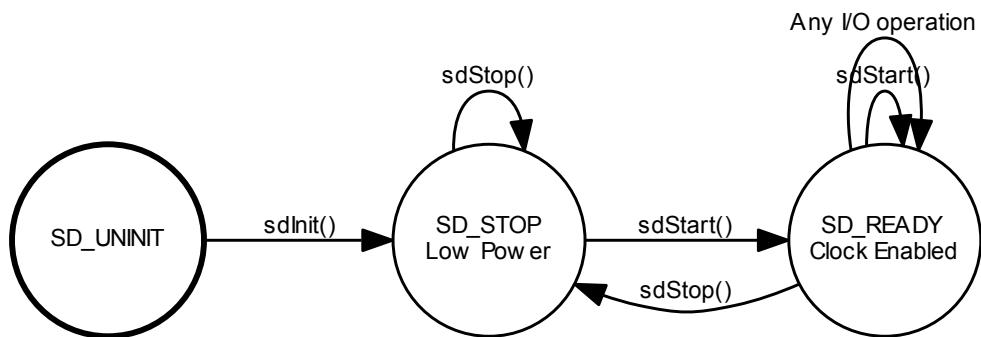
The module also contains functions that make the implementation of the interrupt service routines much easier.

Precondition

In order to use the SERIAL driver the `HAL_USE_SERIAL` option must be enabled in `halconf.h`.

10.48.2 Driver State Machine

The driver implements a state machine internally, not all the driver functionalities can be used in any moment, any transition not explicitly shown in the following diagram has to be considered an error and shall be captured by an assertion (if enabled).



Data Structures

- struct `SerialDriverVMT`
SerialDriver virtual methods table.
- struct `SerialDriver`
Full duplex serial driver class.
- struct `SerialConfig`
Generic Serial Driver configuration structure.

Functions

- void `sdInit` (void)
Serial Driver initialization.
- void `sdObjectInit` (`SerialDriver` *`sdp`, `qnotify_t` `inotify`, `qnotify_t` `onotify`)
Initializes a generic full duplex driver object.
- void `sdStart` (`SerialDriver` *`sdp`, const `SerialConfig` *`config`)
Configures and starts the driver.
- void `sdStop` (`SerialDriver` *`sdp`)
Stops the driver.
- void `sdIncomingData` (`SerialDriver` *`sdp`, `uint8_t` `b`)
Handles incoming data.

- `msg_t sdRequestData (SerialDriver *sdp)`
Handles outgoing data.
- `void sd_lld_init (void)`
Low level serial driver initialization.
- `void sd_lld_start (SerialDriver *sdp, const SerialConfig *config)`
Low level serial driver configuration and (re)start.
- `void sd_lld_stop (SerialDriver *sdp)`
Low level serial driver stop.

Variables

- `SerialDriver SD1`
SD1 driver identifier.

Serial status flags

- `#define SD_PARITY_ERROR 32`
Parity error happened.
- `#define SD_FRAMING_ERROR 64`
Framing error happened.
- `#define SD_OVERRUN_ERROR 128`
Overflow happened.
- `#define SD_NOISE_ERROR 256`
Noise on the line.
- `#define SD_BREAK_DETECTED 512`
Break detected.

Serial configuration options

- `#define SERIAL_DEFAULT_BITRATE 38400`
Default bit rate.
- `#define SERIAL_BUFFERS_SIZE 16`
Serial buffers size.

Macro Functions

- `#define sdPutWouldBlock(sdp) chOQIsFull(&(sdp)->oqueue)`
Direct output check on a `SerialDriver`.
- `#define sdGetWouldBlock(sdp) chIQIsEmpty(&(sdp)->iqueue)`
Direct input check on a `SerialDriver`.
- `#define sdPut(sdp, b) chOQPut(&(sdp)->oqueue, b)`
Direct write to a `SerialDriver`.
- `#define sdPutTimeout(sdp, b, t) chOQPutTimeout(&(sdp)->oqueue, b, t)`
Direct write to a `SerialDriver` with timeout specification.
- `#define sdGet(sdp) chIQGet(&(sdp)->iqueue)`
Direct read from a `SerialDriver`.
- `#define sdGetTimeout(sdp, t) chIQGetTimeout(&(sdp)->iqueue, t)`
Direct read from a `SerialDriver` with timeout specification.
- `#define sdWrite(sdp, b, n) chOQWriteTimeout(&(sdp)->oqueue, b, n, TIME_INFINITE)`
Direct blocking write to a `SerialDriver`.

- `#define sdWriteTimeout(sdp, b, n, t) chOQWriteTimeout(&(sdp)->oqueue, b, n, t)`
Direct blocking write to a [SerialDriver](#) with timeout specification.
- `#define sdAsynchronousWrite(sdp, b, n) chOQWriteTimeout(&(sdp)->oqueue, b, n, TIME_IMMEDIATE)`
Direct non-blocking write to a [SerialDriver](#).
- `#define sdRead(sdp, b, n) chIQReadTimeout(&(sdp)->iqueue, b, n, TIME_INFINITE)`
Direct blocking read from a [SerialDriver](#).
- `#define sdReadTimeout(sdp, b, n, t) chIQReadTimeout(&(sdp)->iqueue, b, n, t)`
Direct blocking read from a [SerialDriver](#) with timeout specification.
- `#define sdAsynchronousRead(sdp, b, n) chIQReadTimeout(&(sdp)->iqueue, b, n, TIME_IMMEDIATE)`
Direct non-blocking read from a [SerialDriver](#).

Configuration options

- `#define PLATFORM_SERIAL_USE_SD1 FALSE`
SD1 driver enable switch.

Defines

- `#define _serial_driver_methods _base_asynchronous_channel_methods`
[SerialDriver](#) specific methods.
- `#define _serial_driver_data`
[SerialDriver](#) specific data.

Typedefs

- `typedef struct SerialDriver SerialDriver`
Structure representing a serial driver.

Enumerations

- `enum sdstate_t { SD_UNINIT = 0, SD_STOP = 1, SD_READY = 2 }`
Driver state machine possible states.

10.48.3 Function Documentation

10.48.3.1 void sdInit (void)

Serial Driver initialization.

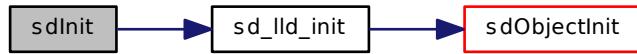
Note

This function is implicitly invoked by `halInit()`, there is no need to explicitly initialize the driver.

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:



10.48.3.2 void sdObjectInit (*SerialDriver* * *sdp*, *qnotify_t* *inotify*, *qnotify_t* *onotify*)

Initializes a generic full duplex driver object.

The HW dependent part of the initialization has to be performed outside, usually in the hardware initialization code.

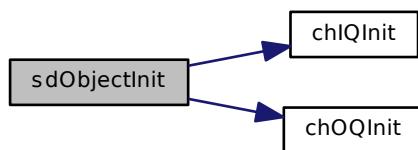
Parameters

<i>out</i>	<i>sdp</i>	pointer to a <i>SerialDriver</i> structure
<i>in</i>	<i>inotify</i>	pointer to a callback function that is invoked when some data is read from the Queue. The value can be <code>NULL</code> .
<i>in</i>	<i>onotify</i>	pointer to a callback function that is invoked when some data is written in the Queue. The value can be <code>NULL</code> .

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:



10.48.3.3 void sdStart (*SerialDriver* * *sdp*, *const SerialConfig* * *config*)

Configures and starts the driver.

Parameters

<i>in</i>	<i>sdp</i>	pointer to a <i>SerialDriver</i> object
<i>in</i>	<i>config</i>	the architecture-dependent serial driver configuration. If this parameter is set to <code>NULL</code> then a default configuration is used.

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:

**10.48.3.4 void sdStop (SerialDriver * sdp)**

Stops the driver.

Any thread waiting on the driver's queues will be awakened with the message Q_RESET.

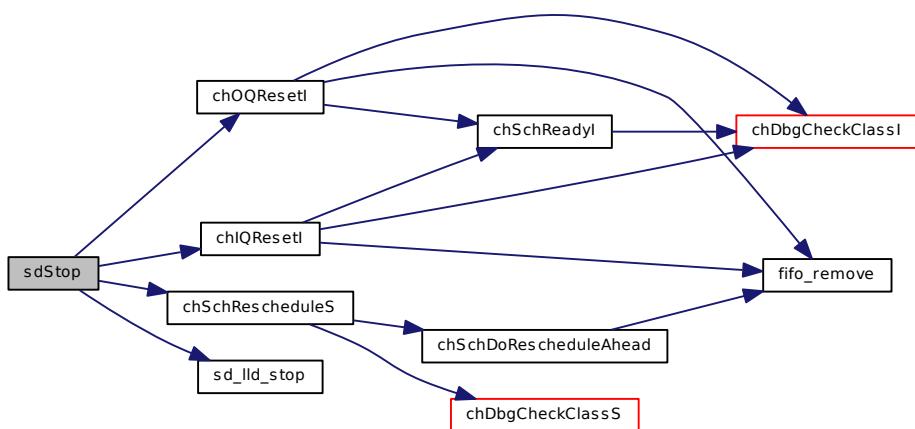
Parameters

in *sdp* pointer to a [SerialDriver](#) 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:

**10.48.3.5 void sdIncomingData (SerialDriver * sdp, uint8_t b)**

Handles incoming data.

This function must be called from the input interrupt service routine in order to enqueue incoming data and generate the related events.

Note

The incoming data event is only generated when the input queue becomes non-empty.

In order to gain some performance it is suggested to not use this function directly but copy this code directly into the interrupt service routine.

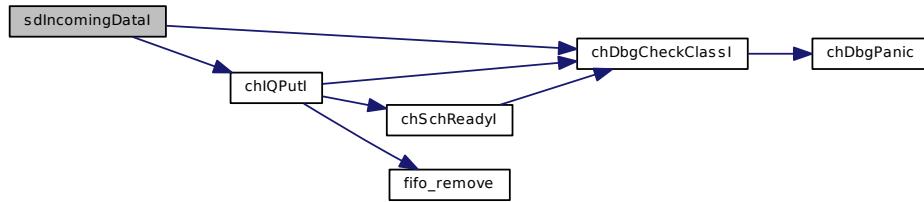
Parameters

in	<i>sdp</i>	pointer to a SerialDriver structure
in	<i>b</i>	the byte to be written in the driver's Input 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:

**10.48.3.6 msg_t sdRequestData (SerialDriver * sdp)**

Handles outgoing data.

Must be called from the output interrupt service routine in order to get the next byte to be transmitted.

Note

In order to gain some performance it is suggested to not use this function directly but copy this code directly into the interrupt service routine.

Parameters

in	<i>sdp</i>	pointer to a SerialDriver structure
----	------------	---

Returns

The byte value read from the driver's output queue.

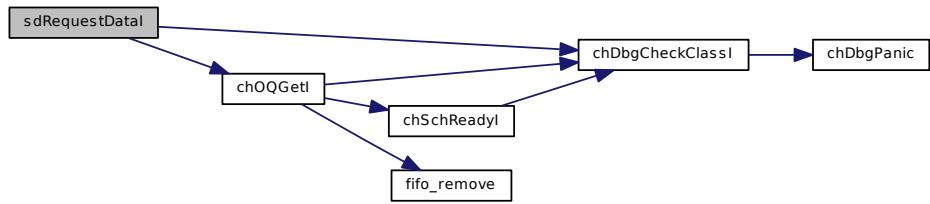
Return values

`Q_EMPTY` if the queue is empty (the lower driver usually disables the interrupt source when this happens).

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:



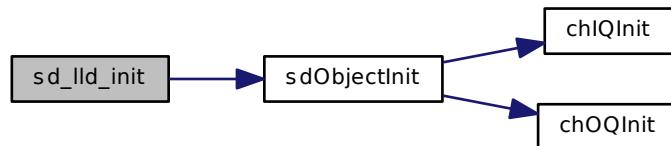
10.48.3.7 void sd_lld_init(void)

Low level serial driver initialization.

Function Class:

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

Here is the call graph for this function:



10.48.3.8 void sd_lld_start(SerialDriver * sdp, const SerialConfig * config)

Low level serial driver configuration and (re)start.

Parameters

in	<i>sdp</i>	pointer to a <code>SerialDriver</code> object
in	<i>config</i>	the architecture-dependent serial driver configuration. If this parameter is set to <code>NULL</code> then a default configuration is used.

Function Class:

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

10.48.3.9 void sd_lld_stop(SerialDriver * sdp)

Low level serial driver stop.

De-initializes the USART, stops the associated clock, resets the interrupt vector.

Parameters

in *sdp* pointer to a `SerialDriver` object

Function Class:

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

10.48.4 Variable Documentation

10.48.4.1 `SerialDriver SD1`

SD1 driver identifier.

10.48.5 Define Documentation

10.48.5.1 `#define SD_PARITY_ERROR 32`

Parity error happened.

10.48.5.2 `#define SD_FRAMING_ERROR 64`

Framing error happened.

10.48.5.3 `#define SD_OVERRUN_ERROR 128`

Overflow happened.

10.48.5.4 `#define SD_NOISE_ERROR 256`

Noise on the line.

10.48.5.5 `#define SD_BREAK_DETECTED 512`

Break detected.

10.48.5.6 `#define SERIAL_DEFAULT_BITRATE 38400`

Default bit rate.

Configuration parameter, this is the baud rate selected for the default configuration.

10.48.5.7 `#define SERIAL_BUFFERS_SIZE 16`

Serial buffers size.

Configuration parameter, you can change the depth of the queue buffers depending on the requirements of your application.

Note

The default is 16 bytes for both the transmission and receive buffers.

```
10.48.5.8 #define _serial_driver_methods _base_asynchronous_channel_methods
```

[SerialDriver](#) specific methods.

```
10.48.5.9 #define sdPutWouldBlock( sdp ) chOQIsFull(&(sdp)->oqueue)
```

Direct output check on a [SerialDriver](#).

Note

This function bypasses the indirect access to the channel and checks directly the output queue. This is faster but cannot be used to check different channels implementations.

Deprecated

Function Class:

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

```
10.48.5.10 #define sdGetWouldBlock( sdp ) chIQIsEmpty(&(sdp)->iqueue)
```

Direct input check on a [SerialDriver](#).

Note

This function bypasses the indirect access to the channel and checks directly the input queue. This is faster but cannot be used to check different channels implementations.

Deprecated

Function Class:

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

```
10.48.5.11 #define sdPut( sdp, b ) chOQPut(&(sdp)->oqueue, b)
```

Direct write to a [SerialDriver](#).

Note

This function bypasses the indirect access to the channel and writes directly on the output queue. This is faster but cannot be used to write to different channels implementations.

See also

[chnPutTimeout\(\)](#)

Function Class:

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

```
10.48.5.12 #define sdPutTimeout( sdp, b, t ) chOQPutTimeout(&(sdp)->oqueue, b, t)
```

Direct write to a [SerialDriver](#) with timeout specification.

Note

This function bypasses the indirect access to the channel and writes directly on the output queue. This is faster but cannot be used to write to different channels implementations.

See also

[chnPutTimeout\(\)](#)

Function Class:

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

```
10.48.5.13 #define sdGet( sdp ) chIQGet(&(sdp)->iqueue)
```

Direct read from a [SerialDriver](#).

Note

This function bypasses the indirect access to the channel and reads directly from the input queue. This is faster but cannot be used to read from different channels implementations.

See also

[chnGetTimeout\(\)](#)

Function Class:

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

```
10.48.5.14 #define sdGetTimeout( sdp, t ) chIQGetTimeout(&(sdp)->iqueue, t)
```

Direct read from a [SerialDriver](#) with timeout specification.

Note

This function bypasses the indirect access to the channel and reads directly from the input queue. This is faster but cannot be used to read from different channels implementations.

See also

[chnGetTimeout\(\)](#)

Function Class:

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

```
10.48.5.15 #define sdWrite( sdp, b, n ) chOQWriteTimeout(&(sdp)->oqueue, b, n, TIME_INFINITE)
```

Direct blocking write to a [SerialDriver](#).

Note

This function bypasses the indirect access to the channel and writes directly to the output queue. This is faster but cannot be used to write from different channels implementations.

See also[chnWrite\(\)](#)**Function Class:**

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

```
10.48.5.16 #define sdWriteTimeout( sdp, b, n, t ) chOQWriteTimeout(&(sdp)->oqueue, b, n, t)
```

Direct blocking write to a [SerialDriver](#) with timeout specification.

Note

This function bypasses the indirect access to the channel and writes directly to the output queue. This is faster but cannot be used to write to different channels implementations.

See also[chnWriteTimeout\(\)](#)**Function Class:**

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

```
10.48.5.17 #define sdAsynchronousWrite( sdp, b, n ) chOQWriteTimeout(&(sdp)->oqueue, b, n, TIME_IMMEDIATE)
```

Direct non-blocking write to a [SerialDriver](#).

Note

This function bypasses the indirect access to the channel and writes directly to the output queue. This is faster but cannot be used to write to different channels implementations.

See also[chnWriteTimeout\(\)](#)**Function Class:**

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

```
10.48.5.18 #define sdRead( sdp, b, n ) chIQReadTimeout(&(sdp)->iqueue, b, n, TIME_INFINITE)
```

Direct blocking read from a [SerialDriver](#).

Note

This function bypasses the indirect access to the channel and reads directly from the input queue. This is faster but cannot be used to read from different channels implementations.

See also[chnRead\(\)](#)**Function Class:**

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

```
10.48.5.19 #define sdReadTimeout( sdp, b, n, t ) chIQReadTimeout(&(sdp)->iqueue, b, n, t)
```

Direct blocking read from a [SerialDriver](#) with timeout specification.

Note

This function bypasses the indirect access to the channel and reads directly from the input queue. This is faster but cannot be used to read from different channels implementations.

See also

[chnReadTimeout\(\)](#)

Function Class:

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

```
10.48.5.20 #define sdAsynchronousRead( sdp, b, n ) chIQReadTimeout(&(sdp)->iqueue, b, n, TIME_IMMEDIATE)
```

Direct non-blocking read from a [SerialDriver](#).

Note

This function bypasses the indirect access to the channel and reads directly from the input queue. This is faster but cannot be used to read from different channels implementations.

See also

[chnReadTimeout\(\)](#)

Function Class:

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

```
10.48.5.21 #define PLATFORM_SERIAL_USE_SD1 FALSE
```

SD1 driver enable switch.

If set to TRUE the support for SD1 is included.

```
10.48.5.22 #define _serial_driver_data
```

Value:

```
_base_asynchronous_channel_data
/* Driver state.*/
sdstate_t           state;
/* Input queue.*/
InputQueue          iqueue;
/* Output queue.*/
OutputQueue         oqueue;
/* Input circular buffer.*/
uint8_t             ib[SERIAL_BUFFERS_SIZE];
/* Output circular buffer.*/
uint8_t             ob[SERIAL_BUFFERS_SIZE];
```

[SerialDriver](#) specific data.

10.48.6 Typedef Documentation

10.48.6.1 `typedef struct SerialDriver SerialDriver`

Structure representing a serial driver.

10.48.7 Enumeration Type Documentation

10.48.7.1 `enum sdstate_t`

Driver state machine possible states.

Enumerator:

SD_UNINIT Not initialized.

SD_STOP Stopped.

SD_READY Ready.

10.49 Serial over USB Driver

10.49.1 Detailed Description

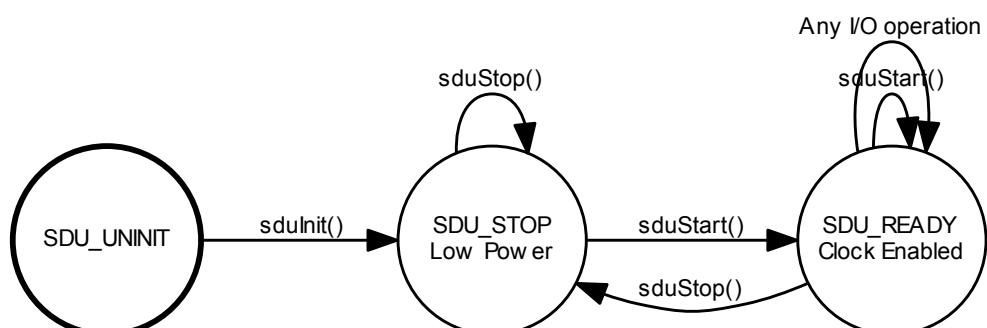
Serial over USB Driver. This module implements an USB Communication Device Class (CDC) as a normal serial communication port accessible from the device application.

Precondition

In order to use the USB over Serial driver the `HAL_USE_SERIAL_USB` option must be enabled in `halconf.h`.

10.49.2 Driver State Machine

The driver implements a state machine internally, not all the driver functionalities can be used in any moment, any transition not explicitly shown in the following diagram has to be considered an error and shall be captured by an assertion (if enabled).



Data Structures

- struct `cdc_linecoding_t`
Type of Line Coding structure.
- struct `SerialUSBConfig`
Serial over USB Driver configuration structure.
- struct `SerialUSBDriverVMT`
SerialDriver virtual methods table.
- struct `SerialUSBDriver`
Full duplex serial driver class.

Functions

- void `sduInit` (void)
Serial Driver initialization.
- void `sduObjectInit` (`SerialUSBDriver` *`sdup`)
Initializes a generic full duplex driver object.
- void `sduStart` (`SerialUSBDriver` *`sdup`, const `SerialUSBConfig` *`config`)
Configures and starts the driver.
- void `sduStop` (`SerialUSBDriver` *`sdup`)
Stops the driver.
- void `sduConfigureHookI` (`SerialUSBDriver` *`sdup`)
USB device configured handler.
- `bool_t sduRequestsHook` (`USBDriver` *`usbp`)
Default requests hook.
- void `sduDataTransmitted` (`USBDriver` *`usbp`, `usbep_t` `ep`)
Default data transmitted callback.
- void `sduDataReceived` (`USBDriver` *`usbp`, `usbep_t` `ep`)
Default data received callback.
- void `sduInterruptTransmitted` (`USBDriver` *`usbp`, `usbep_t` `ep`)
Default data received callback.

CDC specific messages.

- #define `CDC_SEND_ENCAPSULATED_COMMAND` 0x00
- #define `CDC_GET_ENCAPSULATED_RESPONSE` 0x01
- #define `CDC_SET_COMM_FEATURE` 0x02
- #define `CDC_GET_COMM_FEATURE` 0x03
- #define `CDC_CLEAR_COMM_FEATURE` 0x04
- #define `CDC_SET_AUX_LINE_STATE` 0x10
- #define `CDC_SET_HOOK_STATE` 0x11
- #define `CDC_PULSE_SETUP` 0x12
- #define `CDC_SEND_PULSE` 0x13
- #define `CDC_SET_PULSE_TIME` 0x14
- #define `CDC_RING_AUX_JACK` 0x15
- #define `CDC_SET_LINE_CODING` 0x20
- #define `CDC_GET_LINE_CODING` 0x21
- #define `CDC_SET_CONTROL_LINE_STATE` 0x22
- #define `CDC_SEND_BREAK` 0x23
- #define `CDC_SET_RINGER_PARMS` 0x30
- #define `CDC_GET_RINGER_PARMS` 0x31
- #define `CDC_SET_OPERATION_PARMS` 0x32
- #define `CDC_GET_OPERATION_PARMS` 0x33

Line Control bit definitions.

- #define LC_STOP_1 0
- #define LC_STOP_1P5 1
- #define LC_STOP_2 2
- #define LC_PARITY_NONE 0
- #define LC_PARITY_ODD 1
- #define LC_PARITY_EVEN 2
- #define LC_PARITY_MARK 3
- #define LC_PARITY_SPACE 4

SERIAL_USB configuration options

- #define SERIAL_USB_BUFFERS_SIZE 256
Serial over USB buffers size.

Defines

- #define _serial_usb_driver_data
SerialDriver specific data.
- #define _serial_usb_driver_methods _base_asynchronous_channel_methods
SerialUSBDriver specific methods.

Typedefs

- typedef struct SerialUSBDriver SerialUSBDriver
Structure representing a serial over USB driver.

Enumerations

- enum sdustate_t { SDU_UNINIT = 0, SDU_STOP = 1, SDU_READY = 2 }
Driver state machine possible states.

10.49.3 Function Documentation

10.49.3.1 void sdulinit(void)

Serial Driver initialization.

Note

This function is implicitly invoked by `halInit()`, there is no need to explicitly initialize the driver.

Function Class:

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

10.49.3.2 void sduObjectInit(SerialUSBDriver * sdup)

Initializes a generic full duplex driver object.

The HW dependent part of the initialization has to be performed outside, usually in the hardware initialization code.

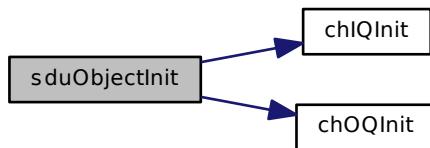
Parameters

out *sduP* pointer to a `SerialUSBDriver` 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:



10.49.3.3 void sduStart (`SerialUSBDriver` * *sduP*, `const SerialUSBConfig` * *config*)

Configures and starts the driver.

Parameters

in	<i>sduP</i>	pointer to a <code>SerialUSBDriver</code> object
in	<i>config</i>	the serial over USB driver configuration

Function Class:

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

10.49.3.4 void sduStop (`SerialUSBDriver` * *sduP*)

Stops the driver.

Any thread waiting on the driver's queues will be awakened with the message `Q_RESET`.

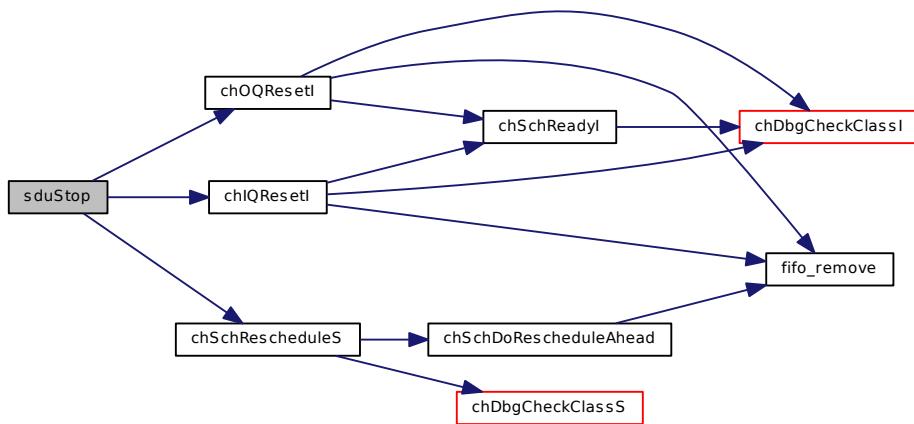
Parameters

in	<i>sduP</i>	pointer to a <code>SerialUSBDriver</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:



10.49.3.5 void sduConfigureHookI (`SerialUSBDriver * sdup`)

USB device configured handler.

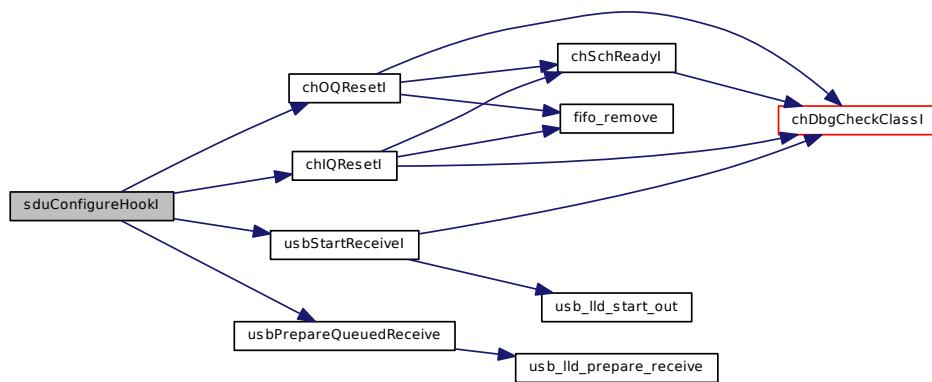
Parameters

in `sdup` pointer to a `SerialUSBDriver` object

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:



10.49.3.6 `bool_t sduRequestsHook (USBDriver * usbp)`

Default requests hook.

Applications wanting to use the Serial over USB driver can use this function as requests hook in the USB configuration. The following requests are emulated:

- `CDC_GET_LINE_CODING`.
- `CDC_SET_LINE_CODING`.
- `CDC_SET_CONTROL_LINE_STATE`.

Parameters

`in` `usbp` pointer to the `USBDriver` object

Returns

The hook status.

Return values

`TRUE` Message handled internally.

`FALSE` Message not handled.

10.49.3.7 `void sduDataTransmitted (USBDriver * usbp, usbep_t ep)`

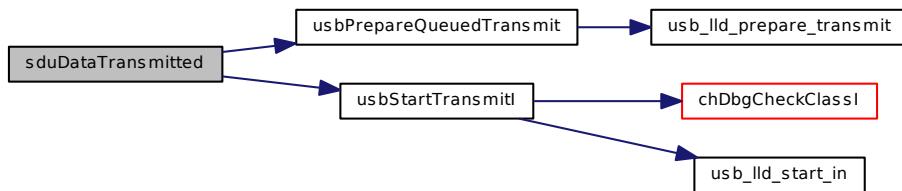
Default data transmitted callback.

The application must use this function as callback for the IN data endpoint.

Parameters

<code>in</code>	<code>usbp</code>	pointer to the <code>USBDriver</code> object
<code>in</code>	<code>ep</code>	endpoint number

Here is the call graph for this function:



10.49.3.8 `void sduDataReceived (USBDriver * usbp, usbep_t ep)`

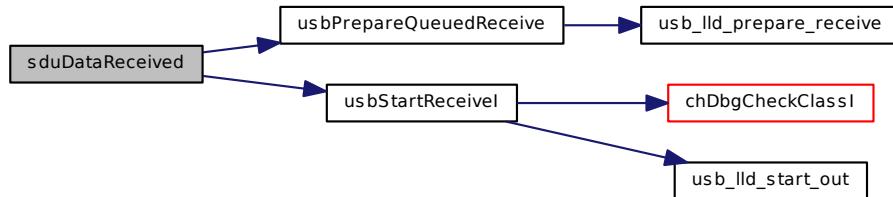
Default data received callback.

The application must use this function as callback for the OUT data endpoint.

Parameters

in	<i>usbp</i>	pointer to the <code>USBDriver</code> object
in	<i>ep</i>	endpoint number

Here is the call graph for this function:



10.49.3.9 void sduInterruptTransmitted(`USBDriver` * *usbp*, `usbep_t` *ep*)

Default data received callback.

The application must use this function as callback for the IN interrupt endpoint.

Parameters

in	<i>usbp</i>	pointer to the <code>USBDriver</code> object
in	<i>ep</i>	endpoint number

10.49.4 Define Documentation

10.49.4.1 #define SERIAL_USB_BUFFERS_SIZE 256

Serial over USB buffers size.

Configuration parameter, the buffer size must be a multiple of the USB data endpoint maximum packet size.

Note

The default is 256 bytes for both the transmission and receive buffers.

10.49.4.2 #define _serial_usb_driver_data

Value:

```

_base_asynchronous_channel_data
/* Driver state.*/
sdustate_t           state;
/* Input queue.*/
InputQueue            iqueue;
/* Output queue.*/
OutputQueue           oqueue;
/* Input buffer.*/
uint8_t                ib[SERIAL_USB_BUFFERS_SIZE];
/* Output buffer.*/
uint8_t                ob[SERIAL_USB_BUFFERS_SIZE];
/* End of the mandatory fields.*/
/* Current configuration data.*/

```

```
const SerialUSBConfig *config;  
  
SerialDriver specific data.
```

10.49.4.3 #define _serial_usb_driver_methods _base_asynchronous_channel_methods

`SerialUSBDriver` specific methods.

10.49.5 Typedef Documentation

10.49.5.1 typedef struct SerialUSBDriver SerialUSBDriver

Structure representing a serial over USB driver.

10.49.6 Enumeration Type Documentation

10.49.6.1 enum sdustate_t

Driver state machine possible states.

Enumerator:

`SDU_UNINIT` Not initialized.

`SDU_STOP` Stopped.

`SDU_READY` Ready.

10.50 SPI Driver

10.50.1 Detailed Description

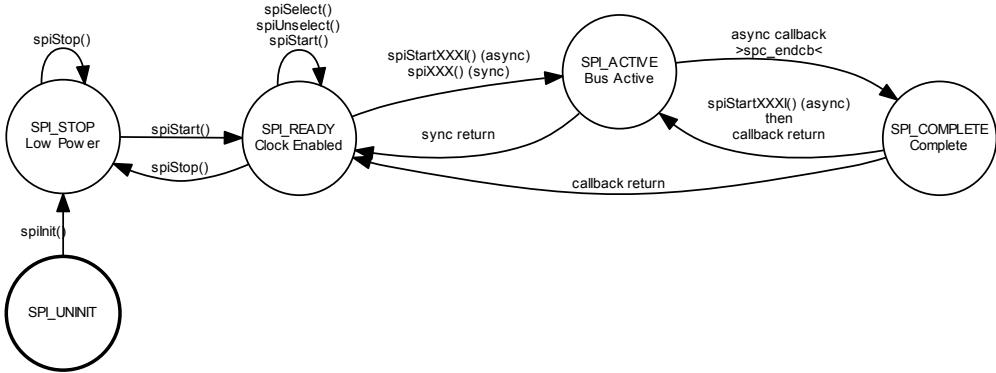
Generic SPI Driver. This module implements a generic SPI (Serial Peripheral Interface) driver allowing bidirectional and monodirectional transfers, complex atomic transactions are supported as well.

Precondition

In order to use the SPI driver the `HAL_USE_SPI` option must be enabled in `halconf.h`.

10.50.2 Driver State Machine

The driver implements a state machine internally, not all the driver functionalities can be used in any moment, any transition not explicitly shown in the following diagram has to be considered an error and shall be captured by an assertion (if enabled).



The driver is not thread safe for performance reasons, if you need to access the SPI bus from multiple threads then use the `spiAcquireBus()` and `spiReleaseBus()` APIs in order to gain exclusive access.

Data Structures

- struct `SPIConfig`
Driver configuration structure.
- struct `SPIDriver`
Structure representing an SPI driver.

Functions

- void `spiInit (void)`
SPI Driver initialization.
- void `spiObjectInit (SPIDriver *spip)`
Initializes the standard part of a `SPIDriver` structure.
- void `spiStart (SPIDriver *spip, const SPIConfig *config)`
Configures and activates the SPI peripheral.
- void `spiStop (SPIDriver *spip)`
Deactivates the SPI peripheral.
- void `spiSelect (SPIDriver *spip)`
Asserts the slave select signal and prepares for transfers.
- void `spiUnselect (SPIDriver *spip)`
Deasserts the slave select signal.
- void `spiStartIgnore (SPIDriver *spip, size_t n)`
Ignores data on the SPI bus.
- void `spiStartExchange (SPIDriver *spip, size_t n, const void *txbuf, void *rxbuf)`
Exchanges data on the SPI bus.
- void `spiStartSend (SPIDriver *spip, size_t n, const void *txbuf)`
Sends data over the SPI bus.
- void `spiStartReceive (SPIDriver *spip, size_t n, void *rxbuf)`
Receives data from the SPI bus.
- void `spignore (SPIDriver *spip, size_t n)`

- `void spiExchange (SPIDriver *spip, size_t n, const void *txbuf, void *rdbuf)`
Exchanges data on the SPI bus.
- `void spiSend (SPIDriver *spip, size_t n, const void *txbuf)`
Sends data over the SPI bus.
- `void spiReceive (SPIDriver *spip, size_t n, void *rdbuf)`
Receives data from the SPI bus.
- `void spiAcquireBus (SPIDriver *spip)`
Gains exclusive access to the SPI bus.
- `void spiReleaseBus (SPIDriver *spip)`
Releases exclusive access to the SPI bus.
- `void spi_lld_init (void)`
Low level SPI driver initialization.
- `void spi_lld_start (SPIDriver *spip)`
Configures and activates the SPI peripheral.
- `void spi_lld_stop (SPIDriver *spip)`
Deactivates the SPI peripheral.
- `void spi_lld_select (SPIDriver *spip)`
Asserts the slave select signal and prepares for transfers.
- `void spi_lld_unselect (SPIDriver *spip)`
Deasserts the slave select signal.
- `void spi_lld_ignore (SPIDriver *spip, size_t n)`
Ignores data on the SPI bus.
- `void spi_lld_exchange (SPIDriver *spip, size_t n, const void *txbuf, void *rdbuf)`
Exchanges data on the SPI bus.
- `void spi_lld_send (SPIDriver *spip, size_t n, const void *txbuf)`
Sends data over the SPI bus.
- `void spi_lld_receive (SPIDriver *spip, size_t n, void *rdbuf)`
Receives data from the SPI bus.
- `uint16_t spi_lld_polled_exchange (SPIDriver *spip, uint16_t frame)`
Exchanges one frame using a polled wait.

Variables

- `SPIDriver SPID1`
SPI1 driver identifier.

SPI configuration options

- `#define SPI_USE_WAIT TRUE`
Enables synchronous APIs.
- `#define SPI_USE_MUTUAL_EXCLUSION TRUE`
Enables the `spiAcquireBus ()` and `spiReleaseBus ()` APIs.

Macro Functions

- `#define spiSelectl(spip)`
Asserts the slave select signal and prepares for transfers.
- `#define spiUnselectl(spip)`
Deasserts the slave select signal.
- `#define spiStartIgnorel(spip, n)`
Ignores data on the SPI bus.
- `#define spiStartExchangel(spip, n, txbuf, rxbuf)`
Exchanges data on the SPI bus.
- `#define spiStartSendl(spip, n, txbuf)`
Sends data over the SPI bus.
- `#define spiStartReceivel(spip, n, rxbuf)`
Receives data from the SPI bus.
- `#define spiPolledExchange(spip, frame) spi_lld_polled_exchange(spip, frame)`
Exchanges one frame using a polled wait.

Low Level driver helper macros

- `#define _spi_wait_s(spip)`
Waits for operation completion.
- `#define _spi_wakeup_isr(spip)`
Wakes up the waiting thread.
- `#define _spi_isr_code(spip)`
Common ISR code.

Configuration options

- `#define PLATFORM_SPI_USE_SPI1 FALSE`
SPI driver enable switch.

Typedefs

- `typedef struct SPIDriver SPIDriver`
Type of a structure representing an SPI driver.
- `typedef void(* spicallback_t)(SPIDriver *spip)`
SPI notification callback type.

Enumerations

- `enum spistate_t {`
`SPI_UNINIT = 0, SPI_STOP = 1, SPI_READY = 2, SPI_ACTIVE = 3,`
`SPI_COMPLETE = 4 }`
Driver state machine possible states.

10.50.3 Function Documentation

10.50.3.1 void spiInit(void)

SPI Driver initialization.

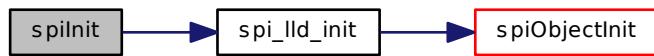
Note

This function is implicitly invoked by `halInit()`, there is no need to explicitly initialize the driver.

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:



10.50.3.2 void spiObjectInit(SPIDriver * spip)

Initializes the standard part of a `SPIDriver` structure.

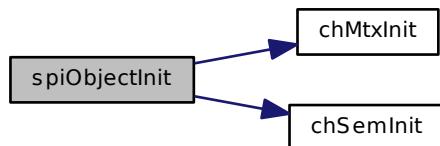
Parameters

`out` `spip` pointer to the `SPIDriver` 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:



10.50.3.3 void spiStart(SPIDriver * spip, const SPIConfig * config)

Configures and activates the SPI peripheral.

Parameters

in	<i>spip</i> pointer to the <code>SPIDriver</code> object
in	<i>config</i> pointer to the <code>SPIConfig</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:

**10.50.3.4 void spiStop (SPIDriver * *spip*)**

Deactivates the SPI peripheral.

Note

Deactivating the peripheral also enforces a release of the slave select line.

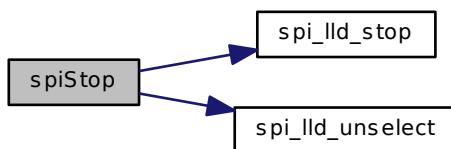
Parameters

in	<i>spip</i> pointer to the <code>SPIDriver</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:

**10.50.3.5 void spiSelect (SPIDriver * *spip*)**

Asserts the slave select signal and prepares for transfers.

Parameters

in *spip* pointer to the `SPIDriver` object

Function Class:

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

10.50.3.6 void spiUnselect (`SPIDriver` * *spip*)

Deasserts the slave select signal.

The previously selected peripheral is unselected.

Parameters

in *spip* pointer to the `SPIDriver` object

Function Class:

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

10.50.3.7 void spiStartIgnore (`SPIDriver` * *spip*, `size_t` *n*)

Ignores data on the SPI bus.

This asynchronous function starts the transmission of a series of idle words on the SPI bus and ignores the received data.

Precondition

A slave must have been selected using `spiSelect()` or `spiSelectI()`.

Postcondition

At the end of the operation the configured callback is invoked.

Parameters

in	<i>spip</i>	pointer to the <code>SPIDriver</code> object
in	<i>n</i>	number of words to be ignored

Function Class:

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

10.50.3.8 void spiStartExchange (`SPIDriver` * *spip*, `size_t` *n*, `const void` * *txbuf*, `void` * *rxbuf*)

Exchanges data on the SPI bus.

This asynchronous function starts a simultaneous transmit/receive operation.

Precondition

A slave must have been selected using `spiSelect()` or `spiSelectI()`.

Postcondition

At the end of the operation the configured callback is invoked.

Note

The buffers are organized as `uint8_t` arrays for data sizes below or equal to 8 bits else it is organized as `uint16_t`

arrays.

Parameters

in	<i>spip</i>	pointer to the <code>SPIDriver</code> object
in	<i>n</i>	number of words to be exchanged
in	<i>txbuf</i>	the pointer to the transmit buffer
out	<i>rxbuf</i>	the pointer to the receive buffer

Function Class:

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

10.50.3.9 void spiStartSend (`SPIDriver * spip`, `size_t n`, `const void * txbuf`)

Sends data over the SPI bus.

This asynchronous function starts a transmit operation.

Precondition

A slave must have been selected using `spiSelect()` or `spiSelectI()`.

Postcondition

At the end of the operation the configured callback is invoked.

Note

The buffers are organized as `uint8_t` arrays for data sizes below or equal to 8 bits else it is organized as `uint16_t` arrays.

Parameters

in	<i>spip</i>	pointer to the <code>SPIDriver</code> object
in	<i>n</i>	number of words to send
in	<i>txbuf</i>	the pointer to the transmit buffer

Function Class:

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

10.50.3.10 void spiStartReceive (`SPIDriver * spip`, `size_t n`, `void * rdbuf`)

Receives data from the SPI bus.

This asynchronous function starts a receive operation.

Precondition

A slave must have been selected using `spiSelect()` or `spiSelectI()`.

Postcondition

At the end of the operation the configured callback is invoked.

Note

The buffers are organized as `uint8_t` arrays for data sizes below or equal to 8 bits else it is organized as `uint16_t` arrays.

Parameters

in	<i>spip</i>	pointer to the <code>SPIDriver</code> object
in	<i>n</i>	number of words to receive
out	<i>rxbuf</i>	the pointer to the receive buffer

Function Class:

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

10.50.3.11 void spiIgnore (`SPIDriver` * *spip*, `size_t` *n*)

Ignores data on the SPI bus.

This synchronous function performs the transmission of a series of idle words on the SPI bus and ignores the received data.

Precondition

In order to use this function the option `SPI_USE_WAIT` must be enabled.

In order to use this function the driver must have been configured without callbacks (`end_cb = NULL`).

Parameters

in	<i>spip</i>	pointer to the <code>SPIDriver</code> object
in	<i>n</i>	number of words to be ignored

Function Class:

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

10.50.3.12 void spiExchange (`SPIDriver` * *spip*, `size_t` *n*, `const void` * *txbuf*, `void` * *rxbuf*)

Exchanges data on the SPI bus.

This synchronous function performs a simultaneous transmit/receive operation.

Precondition

In order to use this function the option `SPI_USE_WAIT` must be enabled.

In order to use this function the driver must have been configured without callbacks (`end_cb = NULL`).

Note

The buffers are organized as `uint8_t` arrays for data sizes below or equal to 8 bits else it is organized as `uint16_t` arrays.

Parameters

in	<i>spip</i>	pointer to the <code>SPIDriver</code> object
in	<i>n</i>	number of words to be exchanged
in	<i>txbuf</i>	the pointer to the transmit buffer
out	<i>rxbuf</i>	the pointer to the receive buffer

Function Class:

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

10.50.3.13 void spiSend (`SPIDriver` * *spip*, `size_t` *n*, `const void` * *txbuf*)

Sends data over the SPI bus.

This synchronous function performs a transmit operation.

Precondition

In order to use this function the option `SPI_USE_WAIT` must be enabled.

In order to use this function the driver must have been configured without callbacks (`end_cb = NULL`).

Note

The buffers are organized as `uint8_t` arrays for data sizes below or equal to 8 bits else it is organized as `uint16_t` arrays.

Parameters

in	<code>spip</code>	pointer to the <code>SPIDriver</code> object
in	<code>n</code>	number of words to send
in	<code>txbuf</code>	the pointer to the transmit buffer

Function Class:

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

10.50.3.14 void spiReceive (`SPIDriver * spip`, `size_t n`, `void * rdbuf`)

Receives data from the SPI bus.

This synchronous function performs a receive operation.

Precondition

In order to use this function the option `SPI_USE_WAIT` must be enabled.

In order to use this function the driver must have been configured without callbacks (`end_cb = NULL`).

Note

The buffers are organized as `uint8_t` arrays for data sizes below or equal to 8 bits else it is organized as `uint16_t` arrays.

Parameters

in	<code>spip</code>	pointer to the <code>SPIDriver</code> object
in	<code>n</code>	number of words to receive
out	<code>rdbuf</code>	the pointer to the receive buffer

Function Class:

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

10.50.3.15 void spiAcquireBus (`SPIDriver * spip`)

Gains exclusive access to the SPI bus.

This function tries to gain ownership to the SPI bus, if the bus is already being used then the invoking thread is queued.

Precondition

In order to use this function the option `SPI_USE_MUTUAL_EXCLUSION` must be enabled.

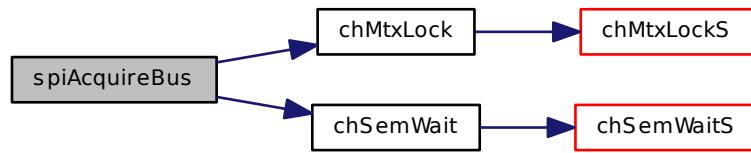
Parameters

in	<code>spip</code>	pointer to the <code>SPIDriver</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:

**10.50.3.16 void spiReleaseBus (SPIDriver * spip)**

Releases exclusive access to the SPI bus.

Precondition

In order to use this function the option `SPI_USE_MUTUAL_EXCLUSION` must be enabled.

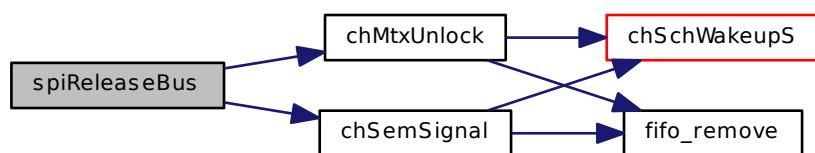
Parameters

in *spip* pointer to the `SPIDriver` 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:

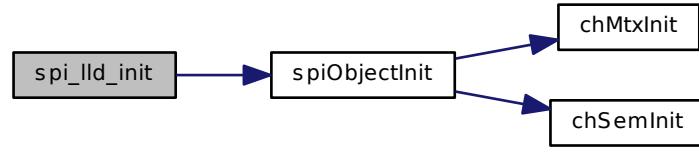
**10.50.3.17 void spi_lld_init (void)**

Low level SPI driver initialization.

Function Class:

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

Here is the call graph for this function:



10.50.3.18 void spi_lld_start (**SPIDriver** * *spip*)

Configures and activates the SPI peripheral.

Parameters

in *spip* pointer to the **SPIDriver** object

Function Class:

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

10.50.3.19 void spi_lld_stop (**SPIDriver** * *spip*)

Deactivates the SPI peripheral.

Parameters

in *spip* pointer to the **SPIDriver** object

Function Class:

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

10.50.3.20 void spi_lld_select (**SPIDriver** * *spip*)

Asserts the slave select signal and prepares for transfers.

Parameters

in *spip* pointer to the **SPIDriver** object

Function Class:

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

10.50.3.21 void spi_lld_unselect (**SPIDriver** * *spip*)

Deasserts the slave select signal.

The previously selected peripheral is unselected.

Parameters

in *spip* pointer to the `SPIDriver` object

Function Class:

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

10.50.3.22 void spi_lld_ignore (`SPIDriver` * *spip*, `size_t` *n*)

Ignores data on the SPI bus.

This asynchronous function starts the transmission of a series of idle words on the SPI bus and ignores the received data.

Postcondition

At the end of the operation the configured callback is invoked.

Parameters

in *spip* pointer to the `SPIDriver` object
in *n* number of words to be ignored

Function Class:

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

10.50.3.23 void spi_lld_exchange (`SPIDriver` * *spip*, `size_t` *n*, const void * *txbuf*, void * *rxbuf*)

Exchanges data on the SPI bus.

This asynchronous function starts a simultaneous transmit/receive operation.

Postcondition

At the end of the operation the configured callback is invoked.

Note

The buffers are organized as `uint8_t` arrays for data sizes below or equal to 8 bits else it is organized as `uint16_t` arrays.

Parameters

in *spip* pointer to the `SPIDriver` object
in *n* number of words to be exchanged
in *txbuf* the pointer to the transmit buffer
out *rxbuf* the pointer to the receive buffer

Function Class:

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

10.50.3.24 void spi_lld_send (`SPIDriver` * *spip*, `size_t` *n*, const void * *txbuf*)

Sends data over the SPI bus.

This asynchronous function starts a transmit operation.

Postcondition

At the end of the operation the configured callback is invoked.

Note

The buffers are organized as uint8_t arrays for data sizes below or equal to 8 bits else it is organized as uint16_t arrays.

Parameters

in	<i>spip</i>	pointer to the SPIDriver object
in	<i>n</i>	number of words to send
in	<i>txbuf</i>	the pointer to the transmit buffer

Function Class:

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

10.50.3.25 void spi_lld_receive (SPIDriver * spip, size_t n, void * rdbuf)

Receives data from the SPI bus.

This asynchronous function starts a receive operation.

Postcondition

At the end of the operation the configured callback is invoked.

Note

The buffers are organized as uint8_t arrays for data sizes below or equal to 8 bits else it is organized as uint16_t arrays.

Parameters

in	<i>spip</i>	pointer to the SPIDriver object
in	<i>n</i>	number of words to receive
out	<i>rdbuf</i>	the pointer to the receive buffer

Function Class:

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

10.50.3.26 uint16_t spi_lld_polled_exchange (SPIDriver * spip, uint16_t frame)

Exchanges one frame using a polled wait.

This synchronous function exchanges one frame using a polled synchronization method. This function is useful when exchanging small amount of data on high speed channels, usually in this situation is much more efficient just wait for completion using polling than suspending the thread waiting for an interrupt.

Parameters

in	<i>spip</i>	pointer to the SPIDriver object
in	<i>frame</i>	the data frame to send over the SPI bus

Returns

The received data frame from the SPI bus.

10.50.4 Variable Documentation

10.50.4.1 **SPIDriver SPID1**

SPI1 driver identifier.

10.50.5 Define Documentation

10.50.5.1 #define SPI_USE_WAIT TRUE

Enables synchronous APIs.

Note

Disabling this option saves both code and data space.

10.50.5.2 #define SPI_USE_MUTUAL_EXCLUSION TRUE

Enables the `spiAcquireBus()` and `spiReleaseBus()` APIs.

Note

Disabling this option saves both code and data space.

10.50.5.3 #define spiSelect(*spip*)

Value:

```
{\n    spi_lld_select(spip);\n}
```

Asserts the slave select signal and prepares for transfers.

Parameters

in *spip* pointer to the `SPIDriver` object

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.

10.50.5.4 #define spiUnselect(*spip*)

Value:

```
{\n    spi_lld_unselect(spip);\n}
```

Deasserts the slave select signal.

The previously selected peripheral is unselected.

Parameters

in *spip* pointer to the `SPIDriver` object

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.

10.50.5.5 `#define spiStartIgnore(spip, n)`

Value:

```
{
    (spip)->state = SPI_ACTIVE;
    spi_lld_ignore(spip, n);
}
```

Ignores data on the SPI bus.

This asynchronous function starts the transmission of a series of idle words on the SPI bus and ignores the received data.

Precondition

A slave must have been selected using `spiSelect()` or `spiSelectI()`.

Postcondition

At the end of the operation the configured callback is invoked.

Parameters

in	<i>spip</i>	pointer to the <code>SPIDriver</code> object
in	<i>n</i>	number of words to be ignored

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.

10.50.5.6 `#define spiStartExchange(spip, n, txbuf, rxbuf)`

Value:

```
{
    (spip)->state = SPI_ACTIVE;
    spi_lld_exchange(spip, n, txbuf, rxbuf);
}
```

Exchanges data on the SPI bus.

This asynchronous function starts a simultaneous transmit/receive operation.

Precondition

A slave must have been selected using `spiSelect()` or `spiSelectI()`.

Postcondition

At the end of the operation the configured callback is invoked.

Note

The buffers are organized as `uint8_t` arrays for data sizes below or equal to 8 bits else it is organized as `uint16_t` arrays.

Parameters

in	<i>spip</i>	pointer to the <code>SPIDriver</code> object
in	<i>n</i>	number of words to be exchanged
in	<i>txbuf</i>	the pointer to the transmit buffer
out	<i>rxbuf</i>	the pointer to the receive buffer

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.

10.50.5.7 #define `spiStartSend()` (*spip*, *n*, *txbuf*)

Value:

```
{
    (spip)->state = SPI_ACTIVE;
    spi_lld_send(spip, n, txbuf);
}
```

Sends data over the SPI bus.

This asynchronous function starts a transmit operation.

Precondition

A slave must have been selected using `spiSelect()` or `spiSelectI()`.

Postcondition

At the end of the operation the configured callback is invoked.

Note

The buffers are organized as `uint8_t` arrays for data sizes below or equal to 8 bits else it is organized as `uint16_t` arrays.

Parameters

in	<i>spip</i>	pointer to the <code>SPIDriver</code> object
in	<i>n</i>	number of words to send
in	<i>txbuf</i>	the pointer to the transmit buffer

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.

10.50.5.8 #define `spiStartReceive()` (*spip*, *n*, *rxbuf*)

Value:

```
{
    (spip)->state = SPI_ACTIVE;
    spi_lld_receive(spip, n, rxbuf);
}
```

Receives data from the SPI bus.

This asynchronous function starts a receive operation.

Precondition

A slave must have been selected using `spiSelect()` or `spiSelectI()`.

Postcondition

At the end of the operation the configured callback is invoked.

Note

The buffers are organized as `uint8_t` arrays for data sizes below or equal to 8 bits else it is organized as `uint16_t` arrays.

Parameters

in	<code>spip</code>	pointer to the <code>SPIDriver</code> object
in	<code>n</code>	number of words to receive
out	<code>rxbuf</code>	the pointer to the receive buffer

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.

10.50.5.9 #define `spiPolledExchange(spip, frame) spi_lld_polled_exchange(spip, frame)`

Exchanges one frame using a polled wait.

This synchronous function exchanges one frame using a polled synchronization method. This function is useful when exchanging small amount of data on high speed channels, usually in this situation is much more efficient just wait for completion using polling than suspending the thread waiting for an interrupt.

Note

This API is implemented as a macro in order to minimize latency.

Parameters

in	<code>spip</code>	pointer to the <code>SPIDriver</code> object
in	<code>frame</code>	the data frame to send over the SPI bus

Returns

The received data frame from the SPI bus.

10.50.5.10 #define `_spi_wait_s(spip)`**Value:**

```
{
    chDbgAssert((spip)->thread == NULL,
                "_spi_wait(), #1", "already waiting");
    (spip)->thread = chThdSelf();
    chSchGoSleepS(THD_STATE_SUSPENDED);
}
```

Waits for operation completion.

This function waits for the driver to complete the current operation.

Precondition

An operation must be running while the function is invoked.

Note

No more than one thread can wait on a SPI driver using this function.

Parameters

in *spip* pointer to the `SPIDriver` object

Function Class:

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

10.50.5.11 #define _spi_wakeup_isr(*spip*)

Value:

```
{
    chSysLockFromIsr();
    if ((spip)->thread != NULL) {
        Thread *tp = (spip)->thread;
        (spip)->thread = NULL;
        tp->p_u.rdymsg = RDY_OK;
        chSchReadyI(tp);
    }
    chSysUnlockFromIsr();
}
```

Wakes up the waiting thread.

Parameters

in *spip* pointer to the `SPIDriver` object

Function Class:

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

10.50.5.12 #define _spi_isr_code(*spip*)

Value:

```
{
    if ((spip)->config->end_cb) {
        (spip)->state = SPI_COMPLETE;
        (spip)->config->end_cb(spip);
        if ((spip)->state == SPI_COMPLETE)
            (spip)->state = SPI_READY;
    }
    else
        (spip)->state = SPI_READY;
    _spi_wakeup_isr(spip);
}
```

Common ISR code.

This code handles the portable part of the ISR code:

- Callback invocation.
- Waiting thread wakeup, if any.
- Driver state transitions.

Note

This macro is meant to be used in the low level drivers implementation only.

Parameters

in *spip* pointer to the [SPIDriver](#) object

Function Class:

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

10.50.5.13 #define PLATFORM_SPI_USE_SPI1 FALSE

SPI driver enable switch.

If set to TRUE the support for SPI1 is included.

10.50.6 Typedef Documentation

10.50.6.1 [typedef struct SPIDriver SPIDriver](#)

Type of a structure representing an SPI driver.

10.50.6.2 [typedef void\(* spicallback_t\)\(SPIDriver *spip\)](#)

SPI notification callback type.

Parameters

in *spip* pointer to the [SPIDriver](#) object triggering the callback

10.50.7 Enumeration Type Documentation

10.50.7.1 [enum spistate_t](#)

Driver state machine possible states.

Enumerator:

SPI_UNINIT Not initialized.

SPI_STOP Stopped.

SPI_READY Ready.

SPI_ACTIVE Exchanging data.

SPI_COMPLETE Asynchronous operation complete.

10.51 Time Measurement Driver

10.51.1 Detailed Description

Time Measurement unit. This module implements a time measurement mechanism able to monitor a portion of code and store the best/worst/last measurement. The measurement is performed using the realtime counter mechanism abstracted in the HAL driver.

Data Structures

- [struct TimeMeasurement](#)

Time Measurement structure.

Functions

- void `tmInit` (void)
Initializes the Time Measurement unit.
- void `tmObjectInit` (`TimeMeasurement` *`tmp`)
Initializes a `TimeMeasurement` object.

Defines

- #define `tmStartMeasurement`(`tmp`) (`tmp`)->`start`(`tmp`)
Starts a measurement.
- #define `tmStopMeasurement`(`tmp`) (`tmp`)->`stop`(`tmp`)
Stops a measurement.

Typedefs

- typedef struct `TimeMeasurement` `TimeMeasurement`
Type of a Time Measurement object.

10.51.2 Function Documentation

10.51.2.1 void `tmInit` (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:



10.51.2.2 void `tmObjectInit` (`TimeMeasurement` * `tmp`)

Initializes a `TimeMeasurement` object.

Parameters

out `tmp` pointer to a `TimeMeasurement` structure

Function Class:

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

10.51.3 Define Documentation

10.51.3.1 #define tmStartMeasurement(*tmp*) (*tmp*)>start(*tmp*)

Starts a measurement.

Precondition

The [TimeMeasurement](#) must be initialized.

Note

This function can be invoked in any context.

Parameters

in, out *tmp* pointer to a [TimeMeasurement](#) structure

Function Class:

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

10.51.3.2 #define tmStopMeasurement(*tmp*) (*tmp*)>stop(*tmp*)

Stops a measurement.

Precondition

The [TimeMeasurement](#) must be initialized.

Note

This function can be invoked in any context.

Parameters

in, out *tmp* pointer to a [TimeMeasurement](#) structure

Function Class:

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

10.51.4 Typedef Documentation

10.51.4.1 typedef struct TimeMeasurement TimeMeasurement

Type of a Time Measurement object.

Note

Start/stop of measurements is performed through the function pointers in order to avoid inlining of those functions which could compromise measurement accuracy.

The maximum measurable time period depends on the implementation of the realtime counter in the HAL driver.

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.

10.52 UART Driver

10.52.1 Detailed Description

Generic UART Driver. This driver abstracts a generic UART (Universal Asynchronous Receiver Transmitter) peripheral, the API is designed to be:

- Unbuffered and copy-less, transfers are always directly performed from/to the application-level buffers without extra copy operations.
- Asynchronous, the API is always non blocking.
- Callbacks capable, operations completion and other events are notified using callbacks.

Special hardware features like deep hardware buffers, DMA transfers are hidden to the user but fully supportable by the low level implementations.

This driver model is best used where communication events are meant to drive an higher level state machine, as example:

- RS485 drivers.
- Multipoint network drivers.
- Serial protocol decoders.

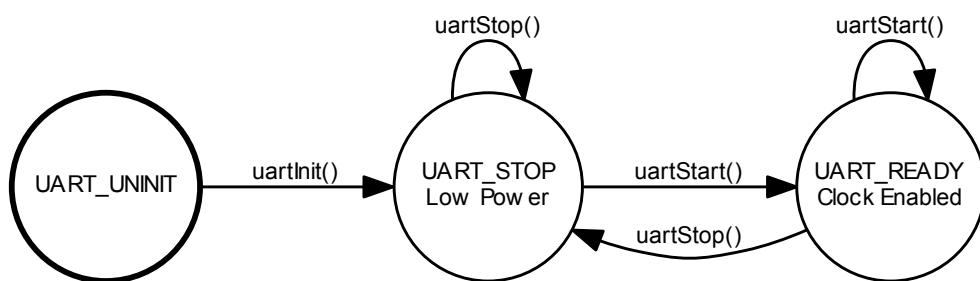
If your application requires a synchronous buffered driver then the [Serial Driver](#) should be used instead.

Precondition

In order to use the UART driver the `HAL_USE_UART` option must be enabled in [halconf.h](#).

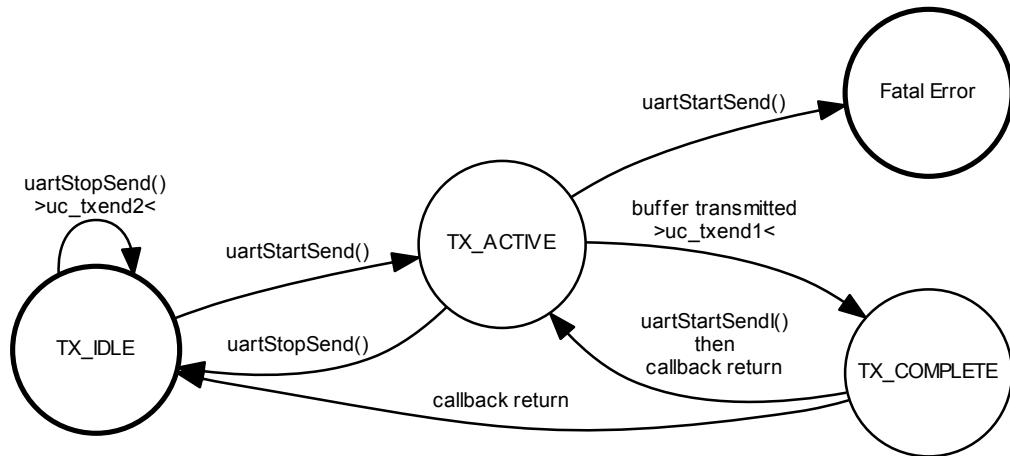
10.52.2 Driver State Machine

The driver implements a state machine internally, not all the driver functionalities can be used in any moment, any transition not explicitly shown in the following diagram has to be considered an error and shall be captured by an assertion (if enabled).



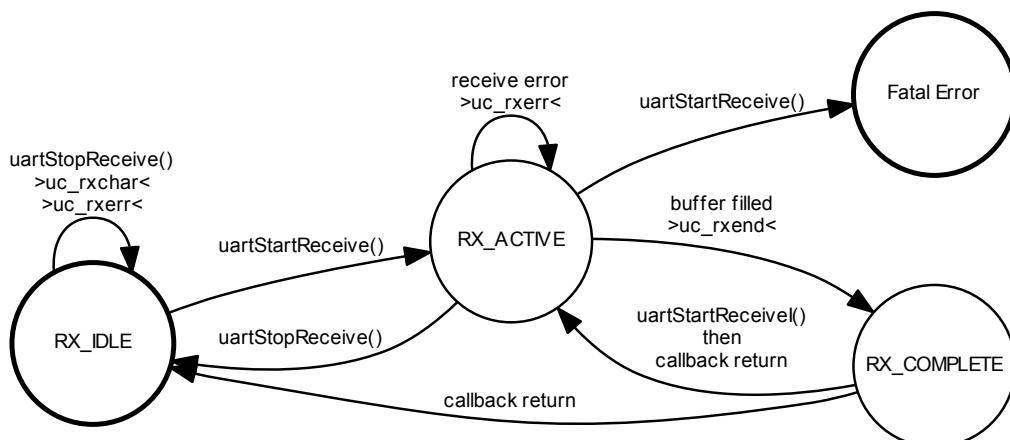
10.52.2.1 Transmitter sub State Machine

The follow diagram describes the transmitter state machine, this diagram is valid while the driver is in the `UART_READY` state. This state machine is automatically reset to the `TX_IDLE` state each time the driver enters the `UART_READY` state.



10.52.2.2 Receiver sub State Machine

The follow diagram describes the receiver state machine, this diagram is valid while the driver is in the `UART_READY` state. This state machine is automatically reset to the `RX_IDLE` state each time the driver enters the `UART_READY` state.



Data Structures

- struct [UARTConfig](#)
Driver configuration structure.
- struct [UARTDriver](#)
Structure representing an UART driver.

Functions

- void `uartInit` (void)
UART Driver initialization.
- void `uartObjectInit` (`UARTDriver` *uartp)
Initializes the standard part of a `UARTDriver` structure.
- void `uartStart` (`UARTDriver` *uartp, const `UARTConfig` *config)
Configures and activates the UART peripheral.
- void `uartStop` (`UARTDriver` *uartp)
Deactivates the UART peripheral.
- void `uartStartSend` (`UARTDriver` *uartp, size_t n, const void *txbuf)
Starts a transmission on the UART peripheral.
- void `uartStartSendl` (`UARTDriver` *uartp, size_t n, const void *txbuf)
Starts a transmission on the UART peripheral.
- size_t `uartStopSend` (`UARTDriver` *uartp)
Stops any ongoing transmission.
- size_t `uartStopSendl` (`UARTDriver` *uartp)
Stops any ongoing transmission.
- void `uartStartReceive` (`UARTDriver` *uartp, size_t n, void *rxbuf)
Starts a receive operation on the UART peripheral.
- void `uartStartReceive1` (`UARTDriver` *uartp, size_t n, void *rxbuf)
Starts a receive operation on the UART peripheral.
- size_t `uartStopReceive` (`UARTDriver` *uartp)
Stops any ongoing receive operation.
- size_t `uartStopReceive1` (`UARTDriver` *uartp)
Stops any ongoing receive operation.
- void `uart_lld_init` (void)
Low level UART driver initialization.
- void `uart_lld_start` (`UARTDriver` *uartp)
Configures and activates the UART peripheral.
- void `uart_lld_stop` (`UARTDriver` *uartp)
Deactivates the UART peripheral.
- void `uart_lld_start_send` (`UARTDriver` *uartp, size_t n, const void *txbuf)
Starts a transmission on the UART peripheral.
- size_t `uart_lld_stop_send` (`UARTDriver` *uartp)
Stops any ongoing transmission.
- void `uart_lld_start_receive` (`UARTDriver` *uartp, size_t n, void *rxbuf)
Starts a receive operation on the UART peripheral.
- size_t `uart_lld_stop_receive` (`UARTDriver` *uartp)
Stops any ongoing receive operation.

Variables

- `UARTDriver` `UARTD1`
UART1 driver identifier.

UART status flags

- `#define UART_NO_ERROR 0`
No pending conditions.
- `#define UART_PARITY_ERROR 4`
Parity error happened.
- `#define UART_FRAMING_ERROR 8`
Framing error happened.
- `#define UART_OVERRUN_ERROR 16`
Overflow happened.
- `#define UART_NOISE_ERROR 32`
Noise on the line.
- `#define UART_BREAK_DETECTED 64`
Break detected.

Configuration options

- `#define PLATFORM_UART_USE_UART1 FALSE`
UART driver enable switch.

Typedefs

- `typedef uint32_t uartflags_t`
UART driver condition flags type.
- `typedef struct UARTDriver UARTDriver`
Type of structure representing an UART driver.
- `typedef void(* uartcb_t)(UARTDriver *uartp)`
Generic UART notification callback type.
- `typedef void(* uartccb_t)(UARTDriver *uartp, uint16_t c)`
Character received UART notification callback type.
- `typedef void(* uarteccb_t)(UARTDriver *uartp, uartflags_t e)`
Receive error UART notification callback type.

Enumerations

- `enum uartstate_t { UART_UNINIT = 0, UART_STOP = 1, UART_READY = 2 }`
Driver state machine possible states.
- `enum uartxstate_t { UART_TX_IDLE = 0, UART_TX_ACTIVE = 1, UART_TX_COMPLETE = 2 }`
Transmitter state machine states.
- `enum uartrxstate_t { UART_RX_IDLE = 0, UART_RX_ACTIVE = 1, UART_RX_COMPLETE = 2 }`
Receiver state machine states.

10.52.3 Function Documentation

10.52.3.1 void uartInit(void)

UART Driver initialization.

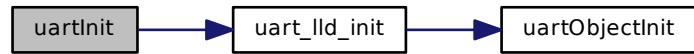
Note

This function is implicitly invoked by `halInit()`, there is no need to explicitly initialize the driver.

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:

**10.52.3.2 void uartObjectInit (**UARTDriver** * *uartp*)**

Initializes the standard part of a **UARTDriver** structure.

Parameters

out	<i>uartp</i> pointer to the UARTDriver object
-----	--

Function Class:

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

10.52.3.3 void uartStart (**UARTDriver * *uartp*, const **UARTConfig** * *config*)**

Configures and activates the UART peripheral.

Parameters

in	<i>uartp</i> pointer to the UARTDriver object
in	<i>config</i> pointer to the UARTConfig 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:

**10.52.3.4 void uartStop (**UARTDriver** * *uartp*)**

Deactivates the UART peripheral.

Parameters

in *uartp* pointer to the [UARTDriver](#) 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:

**10.52.3.5 void uartStartSend ([UARTDriver](#) * *uartp*, *size_t* *n*, const void * *txbuf*)**

Starts a transmission on the UART peripheral.

Note

The buffers are organized as `uint8_t` arrays for data sizes below or equal to 8 bits else it is organized as `uint16_t` arrays.

Parameters

in	<i>uartp</i>	pointer to the UARTDriver object
in	<i>n</i>	number of data frames to send
in	<i>txbuf</i>	the pointer to the transmit buffer

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:

**10.52.3.6 void uartStartSendl ([UARTDriver](#) * *uartp*, *size_t* *n*, const void * *txbuf*)**

Starts a transmission on the UART peripheral.

Note

The buffers are organized as `uint8_t` arrays for data sizes below or equal to 8 bits else it is organized as `uint16_t` arrays.

This function has to be invoked from a lock zone.

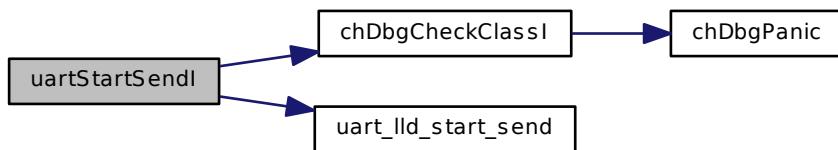
Parameters

in	<i>uartp</i>	pointer to the UARTDriver object
in	<i>n</i>	number of data frames to send
in	<i>txbuf</i>	the pointer to the transmit buffer

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:



10.52.3.7 size_t uartStopSend([UARTDriver](#) * *uartp*)

Stops any ongoing transmission.

Note

Stopping a transmission also suppresses the transmission callbacks.

Parameters

in	<i>uartp</i>	pointer to the UARTDriver object
----	--------------	--

Returns

The number of data frames not transmitted by the stopped transmit operation.

Return values

0 There was no transmit operation in progress.

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:



10.52.3.8 size_t uartStopSend(**UARTDriver** * *uartp*)

Stops any ongoing transmission.

Note

Stopping a transmission also suppresses the transmission callbacks.
This function has to be invoked from a lock zone.

Parameters

in *uartp* pointer to the **UARTDriver** object

Returns

The number of data frames not transmitted by the stopped transmit operation.

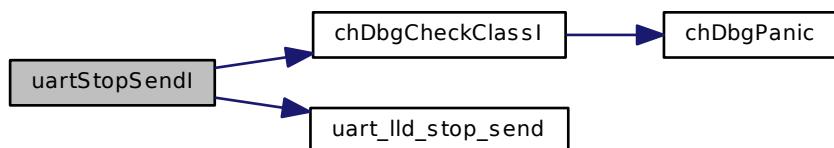
Return values

0 There was no transmit operation in progress.

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:



10.52.3.9 void uartStartReceive(**UARTDriver** * *uartp*, size_t *n*, void * *rdbuf*)

Starts a receive operation on the UART peripheral.

Note

The buffers are organized as `uint8_t` arrays for data sizes below or equal to 8 bits else it is organized as `uint16_t` arrays.

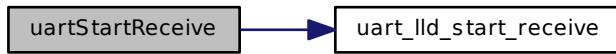
Parameters

in	<code>uartp</code>	pointer to the <code>UARTDriver</code> object
in	<code>n</code>	number of data frames to send
in	<code>rdbuf</code>	the pointer to the receive buffer

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:

**10.52.3.10 void uartStartReceive(`UARTDriver` * `uartp`, `size_t` `n`, `void` * `rdbuf`)**

Starts a receive operation on the UART peripheral.

Note

The buffers are organized as `uint8_t` arrays for data sizes below or equal to 8 bits else it is organized as `uint16_t` arrays.

This function has to be invoked from a lock zone.

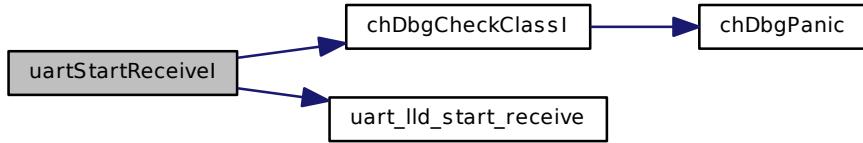
Parameters

in	<code>uartp</code>	pointer to the <code>UARTDriver</code> object
in	<code>n</code>	number of data frames to send
out	<code>rdbuf</code>	the pointer to the receive buffer

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:



10.52.3.11 size_t uartStopReceive (**UARTDriver** * *uartp*)

Stops any ongoing receive operation.

Note

Stopping a receive operation also suppresses the receive callbacks.

Parameters

in *uartp* pointer to the **UARTDriver** object

Returns

The number of data frames not received by the stopped receive operation.

Return values

0 There was no receive operation in progress.

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:



10.52.3.12 size_t uartStopReceive1 (**UARTDriver** * *uartp*)

Stops any ongoing receive operation.

Note

Stopping a receive operation also suppresses the receive callbacks.
This function has to be invoked from a lock zone.

Parameters

in *uartp* pointer to the [UARTDriver](#) object

Returns

The number of data frames not received by the stopped receive operation.

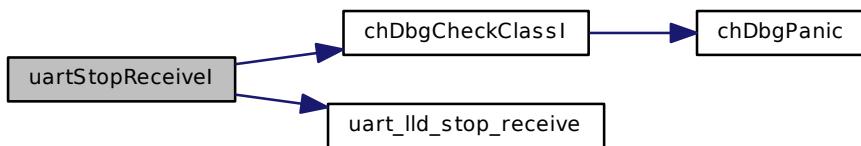
Return values

0 There was no receive operation in progress.

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:

**10.52.3.13 void uart_lld_init(void)**

Low level UART driver initialization.

Function Class:

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

Here is the call graph for this function:

**10.52.3.14 void uart_lld_start([UARTDriver](#) * *uartp*)**

Configures and activates the UART peripheral.

Parameters

in *uartp* pointer to the [UARTDriver](#) object

Function Class:

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

10.52.3.15 void uart_lld_stop (**UARTDriver** * *uartp*)

Deactivates the UART peripheral.

Parameters

in *uartp* pointer to the **UARTDriver** object

Function Class:

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

10.52.3.16 void uart_lld_start_send (**UARTDriver** * *uartp*, **size_t** *n*, const void * *txbuf*)

Starts a transmission on the UART peripheral.

Note

The buffers are organized as **uint8_t** arrays for data sizes below or equal to 8 bits else it is organized as **uint16_t** arrays.

Parameters

in *uartp* pointer to the **UARTDriver** object

in *n* number of data frames to send

in *txbuf* the pointer to the transmit buffer

Function Class:

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

10.52.3.17 **size_t** uart_lld_stop_send (**UARTDriver** * *uartp*)

Stops any ongoing transmission.

Note

Stopping a transmission also suppresses the transmission callbacks.

Parameters

in *uartp* pointer to the **UARTDriver** object

Returns

The number of data frames not transmitted by the stopped transmit operation.

Function Class:

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

10.52.3.18 void uart_lld_start_receive (**UARTDriver** * *uartp*, **size_t** *n*, void * *rxbuf*)

Starts a receive operation on the UART peripheral.

Note

The buffers are organized as `uint8_t` arrays for data sizes below or equal to 8 bits else it is organized as `uint16_t` arrays.

Parameters

in	<i>uartp</i>	pointer to the <code>UARTDriver</code> object
in	<i>n</i>	number of data frames to send
out	<i>rxbuf</i>	the pointer to the receive buffer

Function Class:

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

10.52.3.19 size_t uart_ll_stop_receive (`UARTDriver` * *uartp*)

Stops any ongoing receive operation.

Note

Stopping a receive operation also suppresses the receive callbacks.

Parameters

in	<i>uartp</i>	pointer to the <code>UARTDriver</code> object
----	--------------	---

Returns

The number of data frames not received by the stopped receive operation.

Function Class:

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

10.52.4 Variable Documentation**10.52.4.1 `UARTDriver` `UARTD1`**

UART1 driver identifier.

10.52.5 Define Documentation**10.52.5.1 #define `UART_NO_ERROR` 0**

No pending conditions.

10.52.5.2 #define `UART_PARITY_ERROR` 4

Parity error happened.

10.52.5.3 #define `UART_FRAMING_ERROR` 8

Framing error happened.

10.52.5.4 #define **UART_OVERRUN_ERROR** 16

Overflow happened.

10.52.5.5 #define **UART_NOISE_ERROR** 32

Noise on the line.

10.52.5.6 #define **UART_BREAK_DETECTED** 64

Break detected.

10.52.5.7 #define **PLATFORM_UART_USE_UART1** FALSE

UART driver enable switch.

If set to TRUE the support for UART1 is included.

10.52.6 Typedef Documentation

10.52.6.1 **typedef uint32_t uartflags_t**

UART driver condition flags type.

10.52.6.2 **typedef struct UARTDriver UARTDriver**

Type of structure representing an UART driver.

10.52.6.3 **typedef void(* uartcb_t)(UARTDriver *uartp)**

Generic UART notification callback type.

Parameters

in *uartp* pointer to the **UARTDriver** object

10.52.6.4 **typedef void(* uartccb_t)(UARTDriver *uartp, uint16_t c)**

Character received UART notification callback type.

Parameters

in *uartp* pointer to the **UARTDriver** object triggering the callback

in *c* received character

10.52.6.5 **typedef void(* uartecb_t)(UARTDriver *uartp, uartflags_t e)**

Receive error UART notification callback type.

Parameters

in *uartp* pointer to the **UARTDriver** object triggering the callback

in *e* receive error mask

10.52.7 Enumeration Type Documentation

10.52.7.1 enum uartstate_t

Driver state machine possible states.

Enumerator:

UART_UNINIT Not initialized.

UART_STOP Stopped.

UART_READY Ready.

10.52.7.2 enum uarttxstate_t

Transmitter state machine states.

Enumerator:

UART_TX_IDLE Not transmitting.

UART_TX_ACTIVE Transmitting.

UART_TX_COMPLETE Buffer complete.

10.52.7.3 enum uartrxstate_t

Receiver state machine states.

Enumerator:

UART_RX_IDLE Not receiving.

UART_RX_ACTIVE Receiving.

UART_RX_COMPLETE Buffer complete.

10.53 USB Driver

10.53.1 Detailed Description

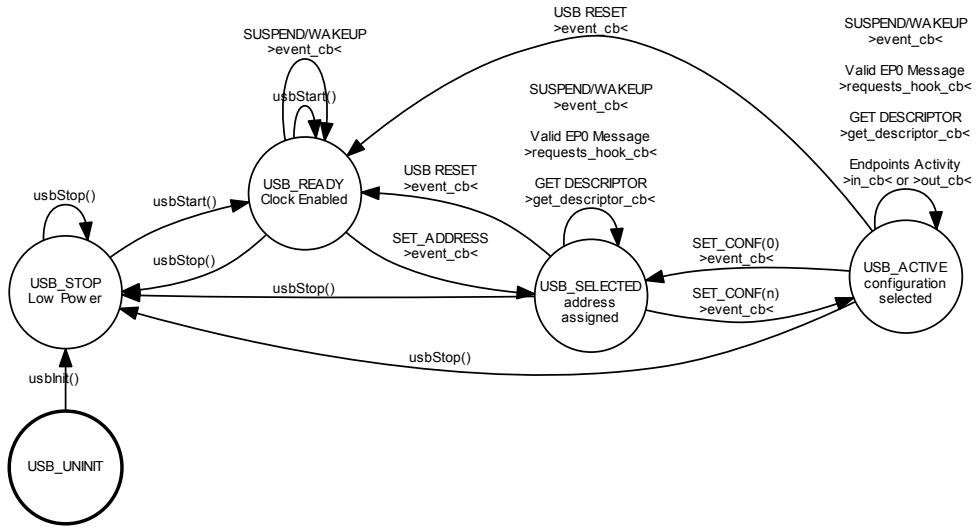
Generic USB Driver. This module implements a generic USB (Universal Serial Bus) driver supporting device-mode operations.

Precondition

In order to use the USB driver the `HAL_USE_USB` option must be enabled in `halconf.h`.

10.53.2 Driver State Machine

The driver implements a state machine internally, not all the driver functionalities can be used in any moment, any transition not explicitly shown in the following diagram has to be considered an error and shall be captured by an assertion (if enabled).



10.53.3 USB Operations

The USB driver is quite complex and USB is complex in itself, it is recommended to study the USB specification before trying to use the driver.

10.53.3.1 USB Implementation

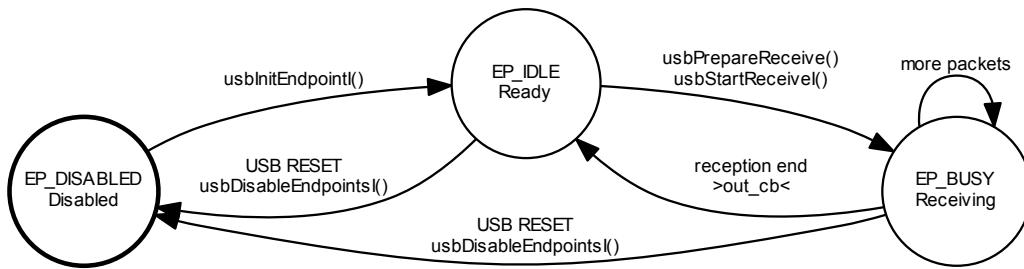
The USB driver abstracts the inner details of the underlying USB hardware. The driver works asynchronously and communicates with the application using callbacks. The application is responsible of the descriptors and strings required by the USB device class to be implemented and of the handling of the specific messages sent over the endpoint zero. Standard messages are handled internally to the driver. The application can use hooks in order to handle custom messages or override the handling of the default handling of standard messages.

10.53.3.2 USB Endpoints

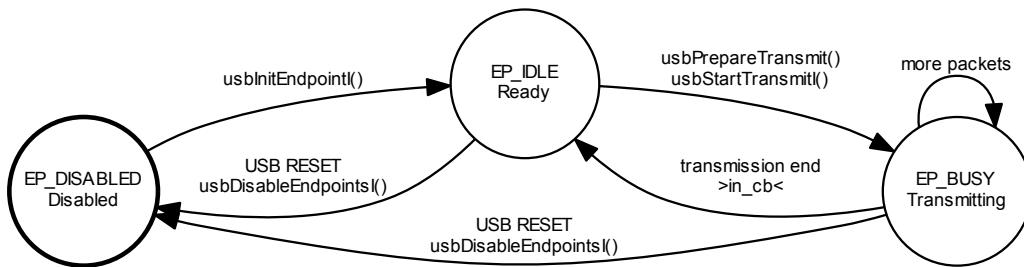
USB endpoints are the objects that the application uses to exchange data with the host. There are two kind of endpoints:

- **IN** endpoints are used by the application to transmit data to the host.
- **OUT** endpoints are used by the application to receive data from the host.

The driver invokes a callback after finishing an IN or OUT transaction. States diagram for OUT endpoints in transaction mode:



States diagram for IN endpoints in transaction mode:



10.53.3.3 USB Callbacks

The USB driver uses callbacks in order to interact with the application. There are several kinds of callbacks to be handled:

- Driver events callback. As example errors, suspend event, reset event etc.
- Messages Hook callback. This hook allows the application to implement handling of custom messages or to override the default handling of standard messages on endpoint zero.
- Descriptor Requested callback. When the driver endpoint zero handler receives a GET DESCRIPTOR message and needs to send a descriptor to the host it queries the application using this callback.
- Start of Frame callback. This callback is invoked each time a SOF packet is received.
- Endpoint callbacks. Each endpoint informs the application about I/O conditions using those callbacks.

Data Structures

- struct [USBDescriptor](#)
Type of an USB descriptor.
- struct [USBInEndpointState](#)
Type of an IN endpoint state structure.
- struct [USBOutEndpointState](#)
Type of an OUT endpoint state structure.

- struct **USBEndpointConfig**
Type of an USB endpoint configuration structure.
- struct **USBConfig**
Type of an USB driver configuration structure.
- struct **USBDriver**
Structure representing an USB driver.

Functions

- void **usbInit** (void)
USB Driver initialization.
- void **usbObjectInit** (**USBDriver** *usbp)
*Initializes the standard part of a **USBDriver** structure.*
- void **usbStart** (**USBDriver** *usbp, const **USBConfig** *config)
Configures and activates the USB peripheral.
- void **usbStop** (**USBDriver** *usbp)
Deactivates the USB peripheral.
- void **usbInitEndpointl** (**USBDriver** *usbp, **usbep_t** ep, const **USBEndpointConfig** *epcp)
Enables an endpoint.
- void **usbDisableEndpointsl** (**USBDriver** *usbp)
Disables all the active endpoints.
- void **usbPrepareReceive** (**USBDriver** *usbp, **usbep_t** ep, **uint8_t** *buf, **size_t** n)
Prepares for a receive transaction on an OUT endpoint.
- void **usbPrepareTransmit** (**USBDriver** *usbp, **usbep_t** ep, const **uint8_t** *buf, **size_t** n)
Prepares for a transmit transaction on an IN endpoint.
- void **usbPrepareQueuedReceive** (**USBDriver** *usbp, **usbep_t** ep, **InputQueue** *iqp, **size_t** n)
Prepares for a receive transaction on an OUT endpoint.
- void **usbPrepareQueuedTransmit** (**USBDriver** *usbp, **usbep_t** ep, **OutputQueue** *oqp, **size_t** n)
Prepares for a transmit transaction on an IN endpoint.
- **bool_t** **usbStartReceivel** (**USBDriver** *usbp, **usbep_t** ep)
Starts a receive transaction on an OUT endpoint.
- **bool_t** **usbStartTransmitl** (**USBDriver** *usbp, **usbep_t** ep)
Starts a transmit transaction on an IN endpoint.
- **bool_t** **usbStallReceivel** (**USBDriver** *usbp, **usbep_t** ep)
Stalls an OUT endpoint.
- **bool_t** **usbStallTransmitl** (**USBDriver** *usbp, **usbep_t** ep)
Stalls an IN endpoint.
- void **_usb_reset** (**USBDriver** *usbp)
USB reset routine.
- void **_usb_ep0setup** (**USBDriver** *usbp, **usbep_t** ep)
Default EP0 SETUP callback.
- void **_usb_ep0in** (**USBDriver** *usbp, **usbep_t** ep)
Default EP0 IN callback.
- void **_usb_ep0out** (**USBDriver** *usbp, **usbep_t** ep)
Default EP0 OUT callback.
- void **usb_lld_init** (void)
Low level USB driver initialization.
- void **usb_lld_start** (**USBDriver** *usbp)
Configures and activates the USB peripheral.
- void **usb_lld_stop** (**USBDriver** *usbp)
Deactivates the USB peripheral.

- void `usb_lld_reset (USBDriver *usbp)`
USB low level reset routine.
- void `usb_lld_set_address (USBDriver *usbp)`
Sets the USB address.
- void `usb_lld_init_endpoint (USBDriver *usbp, usbep_t ep)`
Enables an endpoint.
- void `usb_lld_disable_endpoints (USBDriver *usbp)`
Disables all the active endpoints except the endpoint zero.
- `usbepstatus_t usb_lld_get_status_out (USBDriver *usbp, usbep_t ep)`
Returns the status of an OUT endpoint.
- `usbepstatus_t usb_lld_get_status_in (USBDriver *usbp, usbep_t ep)`
Returns the status of an IN endpoint.
- void `usb_lld_read_setup (USBDriver *usbp, usbep_t ep, uint8_t *buf)`
Reads a setup packet from the dedicated packet buffer.
- void `usb_lld_prepare_receive (USBDriver *usbp, usbep_t ep)`
Prepares for a receive operation.
- void `usb_lld_prepare_transmit (USBDriver *usbp, usbep_t ep)`
Prepares for a transmit operation.
- void `usb_lld_start_out (USBDriver *usbp, usbep_t ep)`
Starts a receive operation on an OUT endpoint.
- void `usb_lld_start_in (USBDriver *usbp, usbep_t ep)`
Starts a transmit operation on an IN endpoint.
- void `usb_lld_stall_out (USBDriver *usbp, usbep_t ep)`
Brings an OUT endpoint in the stalled state.
- void `usb_lld_stall_in (USBDriver *usbp, usbep_t ep)`
Brings an IN endpoint in the stalled state.
- void `usb_lld_clear_out (USBDriver *usbp, usbep_t ep)`
Brings an OUT endpoint in the active state.
- void `usb_lld_clear_in (USBDriver *usbp, usbep_t ep)`
Brings an IN endpoint in the active state.

Variables

- `USBDriver USBD1`
USB1 driver identifier.

Helper macros for USB descriptors

- `#define USB_DESC_INDEX(i) ((uint8_t)(i))`
Helper macro for index values into descriptor strings.
- `#define USB_DESC_BYTE(b) ((uint8_t)(b))`
Helper macro for byte values into descriptor strings.
- `#define USB_DESC_WORD(w)`
Helper macro for word values into descriptor strings.
- `#define USB_DESC_BCD(bcd)`
Helper macro for BCD values into descriptor strings.
- `#define USB_DESC_DEVICE(bcdUSB, bDeviceClass, bDeviceSubClass,bDeviceProtocol, bMaxPacketSize, idVendor,idProduct, bcdDevice, iManufacturer,iProduct, iSerialNumber, bNumConfigurations)`
Device Descriptor helper macro.
- `#define USB_DESC_CONFIGURATION(wTotalLength, bNumInterfaces,bConfigurationValue, iConfiguration,bmAttributes, bMaxPower)`

- Configuration Descriptor helper macro.
• #define **USB_DESC_INTERFACE**(blInterfaceNumber, bAlternateSetting,bNumEndpoints, blInterfaceClass,blInterfaceSubClass, blInterfaceProtocol,ilInterface)
- Interface Descriptor helper macro.
- #define **USB_DESC_INTERFACE_ASSOCIATION**(bFirstInterface,blInterfaceCount, bFunctionClass,bFunctionSubClass, bFunctionProcotol,ilInterface)
- Interface Association Descriptor helper macro.
- #define **USB_DESC_ENDPOINT**(bEndpointAddress, bmAttributes, wMaxPacketSize,blInterval)
- Endpoint Descriptor helper macro.

Endpoint types and settings

- #define **USB_EP_MODE_TYPE** 0x0003
- #define **USB_EP_MODE_TYPE_CTRL** 0x0000
- #define **USB_EP_MODE_TYPE_ISOC** 0x0001
- #define **USB_EP_MODE_TYPE_BULK** 0x0002
- #define **USB_EP_MODE_TYPE_INTR** 0x0003
- #define **USB_EP_MODE_LINEAR_BUFFER** 0x0000
- #define **USB_EP_MODE_QUEUE_BUFFER** 0x0010

Macro Functions

- #define **usbGetDriverState**(usbp) ((usbp)->state)
 Returns the driver state.
- #define **usbFetchWord**(p) ((uint16_t)*(p) | ((uint16_t)*((p) + 1) << 8))
 Fetches a 16 bits word value from an USB message.
- #define **usbConnectBus**(usbp) usb_lld_connect_bus(usbp)
 Connects the USB device.
- #define **usbDisconnectBus**(usbp) usb_lld_disconnect_bus(usbp)
 Disconnect the USB device.
- #define **usbGetFrameNumber**(usbp) usb_lld_get_frame_number(usbp)
 Returns the current frame number.
- #define **usbGetTransmitStatus**(usbp, ep) ((usbp)->transmitting & (1 << (ep)))
 Returns the status of an IN endpoint.
- #define **usbGetReceiveStatus**(usbp, ep) ((usbp)->receiving & (1 << (ep)))
 Returns the status of an OUT endpoint.
- #define **usbGetReceiveTransactionSize**(usbp, ep) usb_lld_get_transaction_size(usbp, ep)
 Returns the exact size of a receive transaction.
- #define **usbSetupTransfer**(usbp, buf, n, endcb)
 Request transfer setup.
- #define **usbReadSetup**(usbp, ep, buf) usb_lld_read_setup(usbp, ep, buf)
 Reads a setup packet from the dedicated packet buffer.

Low Level driver helper macros

- #define **_usb_isr_invoke_event_cb**(usbp, evt)
 Common ISR code, usb event callback.
- #define **_usb_isr_invoke_sof_cb**(usbp)
 Common ISR code, SOF callback.
- #define **_usb_isr_invoke_setup_cb**(usbp, ep)
 Common ISR code, setup packet callback.

- `#define _usb_isr_invoke_in_cb(usbp, ep)`
Common ISR code, IN endpoint callback.
- `#define _usb_isr_invoke_out_cb(usbp, ep)`
Common ISR code, OUT endpoint event.

Configuration options

- `#define PLATFORM_USB_USE_USB1 FALSE`
USB driver enable switch.

Defines

- `#define USB_MAX_ENDPOINTS 4`
Maximum endpoint address.
- `#define USB_EP0_STATUS_STAGE USB_EP0_STATUS_STAGE_SW`
Status stage handling method.
- `#define USB_SET_ADDRESS_MODE USB_EARLY_SET_ADDRESS`
The address can be changed immediately upon packet reception.
- `#define USB_SET_ADDRESS_ACK_HANDLING USB_SET_ADDRESS_ACK_SW`
Method for set address acknowledge.
- `#define usb_ll_get_transaction_size(usbp, ep) ((usbp)->epc[ep]->out_state->rxcnt)`
Returns the exact size of a receive transaction.
- `#define usb_ll_connect_bus(usbp)`
Connects the USB device.
- `#define usb_ll_disconnect_bus(usbp)`
Disconnect the USB device.

Typedefs

- `typedef struct USBDriver USBDriver`
Type of a structure representing an USB driver.
- `typedef uint8_t usbep_t`
Type of an endpoint identifier.
- `typedef void(* usbcallback_t)(USBDriver *usbp)`
Type of an USB generic notification callback.
- `typedef void(* usbepcallback_t)(USBDriver *usbp, usbep_t ep)`
Type of an USB endpoint callback.
- `typedef void(* usbeventcb_t)(USBDriver *usbp, usbevent_t event)`
Type of an USB event notification callback.
- `typedef bool_t(* usbreqhandler_t)(USBDriver *usbp)`
Type of a requests handler callback.
- `typedef const USBDescriptor *(* usbgetdescriptor_t)(USBDriver *usbp, uint8_t dtype, uint8_t dindex, uint16_t lang)`
Type of an USB descriptor-retrieving callback.

Enumerations

- enum `usbstate_t` {

`USB_UNINIT` = 0, `USB_STOP` = 1, `USB_READY` = 2, `USB_SELECTED` = 3,

`USB_ACTIVE` = 4 }

Type of a driver state machine possible states.
- enum `usbepstatus_t` { `EP_STATUS_DISABLED` = 0, `EP_STATUS_STALLED` = 1, `EP_STATUS_ACTIVE` = 2 }

Type of an endpoint status.
- enum `usbep0state_t` {

`USB_EP0_WAITING_SETUP`, `USB_EP0_TX`, `USB_EP0_WAITING_TX0`, `USB_EP0_WAITING_STS`,

`USB_EP0_RX`, `USB_EP0_SENDING_STS`, `USB_EP0_ERROR` }

Type of an endpoint zero state machine states.
- enum `usbevent_t` {

`USB_EVENT_RESET` = 0, `USB_EVENT_ADDRESS` = 1, `USB_EVENT_CONFIGURED` = 2, `USB_EVENT_SUSPEND` = 3,

`USB_EVENT_WAKEUP` = 4, `USB_EVENT_STALLED` = 5 }

Type of an enumeration of the possible USB events.

10.53.4 Function Documentation

10.53.4.1 void usbInit(void)

USB Driver initialization.

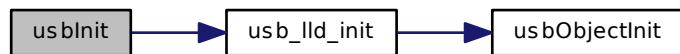
Note

This function is implicitly invoked by `halInit()`, there is no need to explicitly initialize the driver.

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:



10.53.4.2 void usObjectInit(USBDriver * usb)

Initializes the standard part of a `USBDriver` structure.

Parameters

`out` `usb` pointer to the `USBDriver` object

Function Class:

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

10.53.4.3 void usbStart (**USBDriver * *usbp*, const **USBConfig** * *config*)**

Configures and activates the USB peripheral.

Parameters

in	<i>usbp</i>	pointer to the USBDriver object
in	<i>config</i>	pointer to the USBConfig 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:

**10.53.4.4 void usbStop (**USBDriver** * *usbp*)**

Deactivates the USB peripheral.

Parameters

in	<i>usbp</i>	pointer to the USBDriver 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:

**10.53.4.5 void usblInitEndpointl (**USBDriver** * *usbp*, **usbep_t** *ep*, const **USBEndpointConfig** * *epcp*)**

Enables an endpoint.

This function enables an endpoint, both IN and/or OUT directions depending on the configuration structure.

Note

This function must be invoked in response of a SET_CONFIGURATION or SET_INTERFACE message.

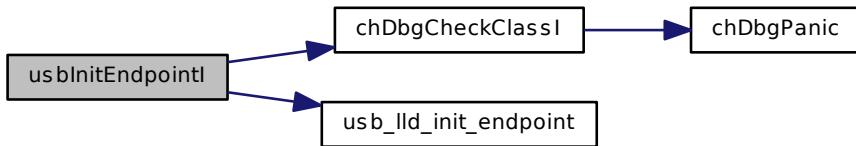
Parameters

in	<i>usbp</i>	pointer to the USBDriver object
in	<i>ep</i>	endpoint number
in	<i>epcp</i>	the endpoint configuration

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:



10.53.4.6 void usbDisableEndpoints1 ([USBDriver](#) * *usbp*)

Disables all the active endpoints.

This function disables all the active endpoints except the endpoint zero.

Note

This function must be invoked in response of a SET_CONFIGURATION message with configuration number zero.

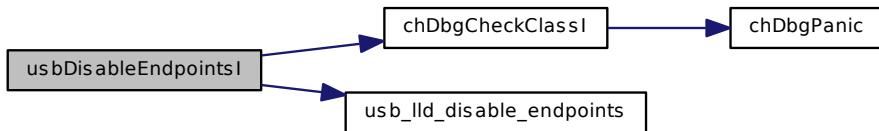
Parameters

in	<i>usbp</i>	pointer to the USBDriver object
----	-------------	---

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:



10.53.4.7 void usbPrepareReceive (*USBDriver* * *usbp*, *usbep_t* *ep*, *uint8_t* * *buf*, *size_t* *n*)

Prepares for a receive transaction on an OUT endpoint.

Postcondition

The endpoint is ready for [usbStartReceiveI\(\)](#).

Note

This function can be called both in ISR and thread context.

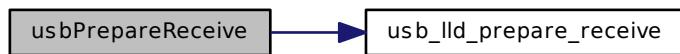
Parameters

in	<i>usbp</i>	pointer to the USBDriver object
in	<i>ep</i>	endpoint number
out	<i>buf</i>	buffer where to copy the received data
in	<i>n</i>	transaction size

Function Class:

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

Here is the call graph for this function:



10.53.4.8 void usbPrepareTransmit (*USBDriver* * *usbp*, *usbep_t* *ep*, *const uint8_t* * *buf*, *size_t* *n*)

Prepares for a transmit transaction on an IN endpoint.

Postcondition

The endpoint is ready for [usbStartTransmitI\(\)](#).

Note

This function can be called both in ISR and thread context.
The queue must contain at least the amount of data specified as transaction size.

Parameters

in	<i>usbp</i>	pointer to the USBDriver object
in	<i>ep</i>	endpoint number
in	<i>buf</i>	buffer where to fetch the data to be transmitted
in	<i>n</i>	transaction size

Function Class:

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

Here is the call graph for this function:



10.53.4.9 void usbPrepareQueuedReceive ([USBDriver](#) * *usbp*, [usbep_t](#) *ep*, [InputQueue](#) * *iqp*, [size_t](#) *n*)

Prepares for a receive transaction on an OUT endpoint.

Postcondition

The endpoint is ready for [usbStartReceiveI\(\)](#).

Note

This function can be called both in ISR and thread context.
The queue must have enough free space to accommodate the specified transaction size rounded to the next packet size boundary. For example if the transaction size is 1 and the packet size is 64 then the queue must have space for at least 64 bytes.

Parameters

in	<i>usbp</i>	pointer to the USBDriver object
in	<i>ep</i>	endpoint number
in	<i>iqp</i>	input queue to be filled with incoming data
in	<i>n</i>	transaction size

Function Class:

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

Here is the call graph for this function:



10.53.4.10 void usbPrepareQueuedTransmit (**USBDriver** * *usbp*, **usbep_t** *ep*, **OutputQueue** * *oqp*, **size_t** *n*)

Prepares for a transmit transaction on an IN endpoint.

Postcondition

The endpoint is ready for [usbStartTransmitI\(\)](#).

Note

This function can be called both in ISR and thread context.

The transmit transaction size is equal to the data contained in the queue.

Parameters

in	<i>usbp</i>	pointer to the USBDriver object
in	<i>ep</i>	endpoint number
in	<i>oqp</i>	output queue to be fetched for outgoing data
in	<i>n</i>	transaction size

Function Class:

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

Here is the call graph for this function:



10.53.4.11 **bool_t** usbStartReceiv1 (**USBDriver** * *usbp*, **usbep_t** *ep*)

Starts a receive transaction on an OUT endpoint.

Postcondition

The endpoint callback is invoked when the transfer has been completed.

Parameters

in	<i>usbp</i>	pointer to the <code>USBDriver</code> object
in	<i>ep</i>	endpoint number

Returns

The operation status.

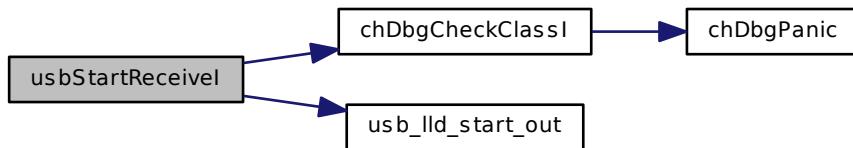
Return values

<i>FALSE</i>	Operation started successfully.
<i>TRUE</i>	Endpoint busy, operation not started.

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:

**10.53.4.12 bool_t usbStartTransmit(USBDriver * *usbp*, usbep_t *ep*)**

Starts a transmit transaction on an IN endpoint.

Postcondition

The endpoint callback is invoked when the transfer has been completed.

Parameters

in	<i>usbp</i>	pointer to the <code>USBDriver</code> object
in	<i>ep</i>	endpoint number

Returns

The operation status.

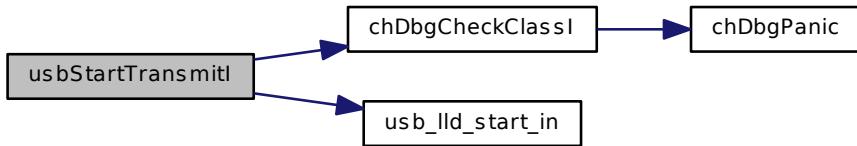
Return values

<i>FALSE</i>	Operation started successfully.
<i>TRUE</i>	Endpoint busy, operation not started.

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:



10.53.4.13 bool_t usbStallReceive(USBDriver * usbp, usbep_t ep)

Stalls an OUT endpoint.

Parameters

in	<i>usbp</i>	pointer to the <code>USBDriver</code> object
in	<i>ep</i>	endpoint number

Returns

The operation status.

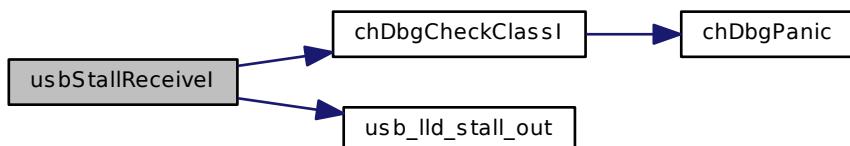
Return values

<i>FALSE</i>	Endpoint stalled.
<i>TRUE</i>	Endpoint busy, not stalled.

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:



10.53.4.14 bool_t usbStallTransmit(USBDriver * usbp, usbep_t ep)

Stalls an IN endpoint.

Parameters

in	<i>usbp</i> pointer to the <code>USBDriver</code> object
in	<i>ep</i> endpoint number

Returns

The operation status.

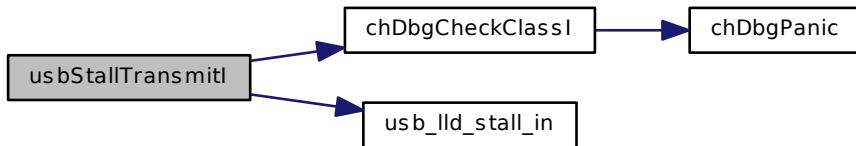
Return values

<i>FALSE</i>	Endpoint stalled.
<i>TRUE</i>	Endpoint busy, not stalled.

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:

**10.53.4.15 void _usb_reset (`USBDriver` * *usbp*)**

USB reset routine.

This function must be invoked when an USB bus reset condition is detected.

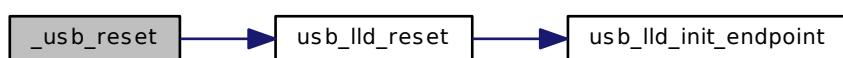
Parameters

in	<i>usbp</i> pointer to the <code>USBDriver</code> object
----	--

Function Class:

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

Here is the call graph for this function:



10.53.4.16 void _usb_ep0setup (**USBDriver** * *usbp*, **usbep_t** *ep*)

Default EP0 SETUP callback.

This function is used by the low level driver as default handler for EP0 SETUP events.

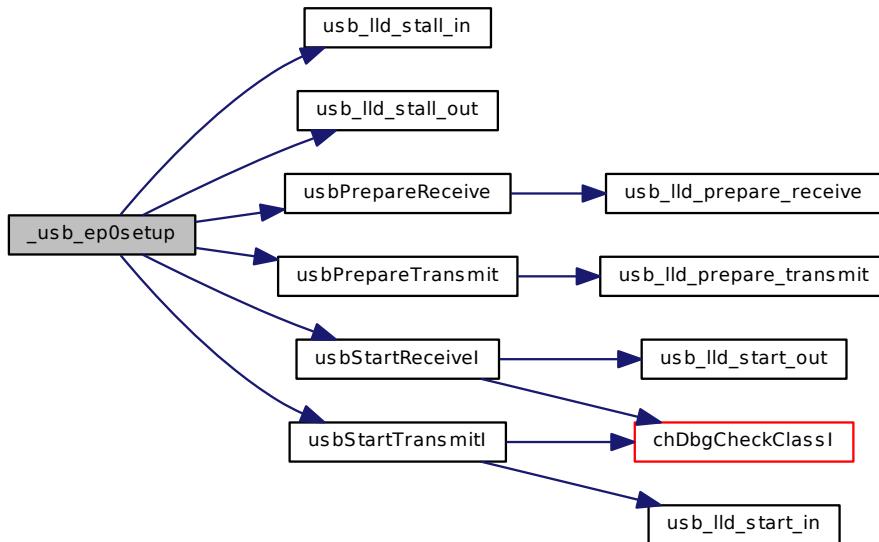
Parameters

in	<i>usbp</i>	pointer to the USBDriver object
in	<i>ep</i>	endpoint number, always zero

Function Class:

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

Here is the call graph for this function:



10.53.4.17 void _usb_ep0in (**USBDriver** * *usbp*, **usbep_t** *ep*)

Default EP0 IN callback.

This function is used by the low level driver as default handler for EP0 IN events.

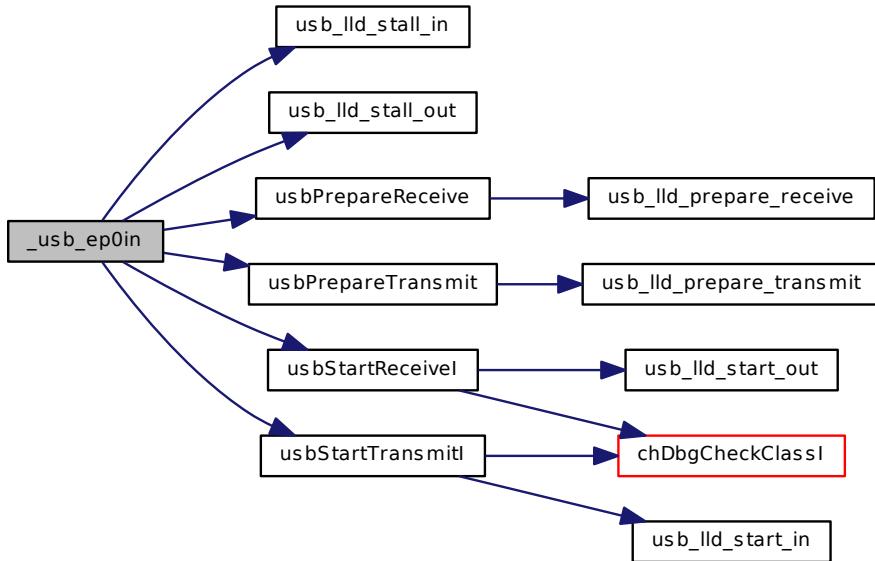
Parameters

in	<i>usbp</i>	pointer to the USBDriver object
in	<i>ep</i>	endpoint number, always zero

Function Class:

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

Here is the call graph for this function:



10.53.4.18 void _usb_ep0out(USBDriver * *usbp*, usbep_t *ep*)

Default EP0 OUT callback.

This function is used by the low level driver as default handler for EP0 OUT events.

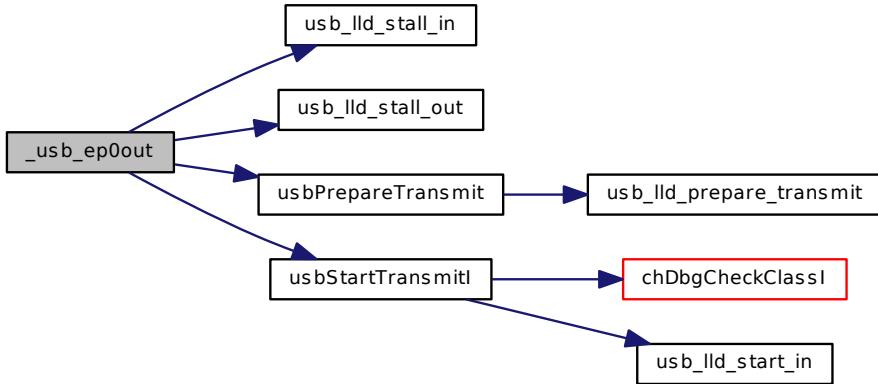
Parameters

in	<i>usbp</i>	pointer to the <code>USBDriver</code> object
in	<i>ep</i>	endpoint number, always zero

Function Class:

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

Here is the call graph for this function:



10.53.4.19 void usb_lld_init(void)

Low level USB driver initialization.

Function Class:

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

Here is the call graph for this function:



10.53.4.20 void usb_lld_start(USBDriver * usbp)

Configures and activates the USB peripheral.

Parameters

`in` `usbp` pointer to the `USBDriver` object

Function Class:

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

10.53.4.21 void usb_lld_stop (**USBDriver** * *usbp*)

Deactivates the USB peripheral.

Parameters

in *usbp* pointer to the **USBDriver** object

Function Class:

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

10.53.4.22 void usb_lld_reset (**USBDriver** * *usbp*)

USB low level reset routine.

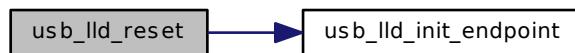
Parameters

in *usbp* pointer to the **USBDriver** object

Function Class:

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

Here is the call graph for this function:



10.53.4.23 void usb_lld_set_address (**USBDriver** * *usbp*)

Sets the USB address.

Parameters

in *usbp* pointer to the **USBDriver** object

Function Class:

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

10.53.4.24 void usb_lld_init_endpoint (**USBDriver** * *usbp*, **usbep_t** *ep*)

Enables an endpoint.

Parameters

in *usbp* pointer to the **USBDriver** object

in *ep* endpoint number

Function Class:

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

10.53.4.25 void usb_lld_disable_endpoints (**USBDriver * *usbp*)**

Disables all the active endpoints except the endpoint zero.

Parameters

in *usbp* pointer to the **USBDriver** object

Function Class:

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

10.53.4.26 usbepstatus_t usb_lld_get_status_out (**USBDriver * *usbp*, **usbep_t** *ep*)**

Returns the status of an OUT endpoint.

Parameters

in *usbp* pointer to the **USBDriver** object
in *ep* endpoint number

Returns

The endpoint status.

Return values

EP_STATUS_DISABLED The endpoint is not active.
EP_STATUS_STALLED The endpoint is stalled.
EP_STATUS_ACTIVE The endpoint is active.

Function Class:

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

10.53.4.27 usbepstatus_t usb_lld_get_status_in (**USBDriver * *usbp*, **usbep_t** *ep*)**

Returns the status of an IN endpoint.

Parameters

in *usbp* pointer to the **USBDriver** object
in *ep* endpoint number

Returns

The endpoint status.

Return values

EP_STATUS_DISABLED The endpoint is not active.
EP_STATUS_STALLED The endpoint is stalled.
EP_STATUS_ACTIVE The endpoint is active.

Function Class:

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

10.53.4.28 void usb_lld_read_setup (**USBDriver** * *usbp*, **usbep_t** *ep*, **uint8_t** * *buf*)

Reads a setup packet from the dedicated packet buffer.

This function must be invoked in the context of the `setup_cb` callback in order to read the received setup packet.

Precondition

In order to use this function the endpoint must have been initialized as a control endpoint.

Postcondition

The endpoint is ready to accept another packet.

Parameters

in	<i>usbp</i>	pointer to the <code>USBDriver</code> object
in	<i>ep</i>	endpoint number
out	<i>buf</i>	buffer where to copy the packet data

Function Class:

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

10.53.4.29 void usb_lld_prepare_receive (**USBDriver** * *usbp*, **usbep_t** *ep*)

Prepares for a receive operation.

Parameters

in	<i>usbp</i>	pointer to the <code>USBDriver</code> object
in	<i>ep</i>	endpoint number

Function Class:

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

10.53.4.30 void usb_lld_prepare_transmit (**USBDriver** * *usbp*, **usbep_t** *ep*)

Prepares for a transmit operation.

Parameters

in	<i>usbp</i>	pointer to the <code>USBDriver</code> object
in	<i>ep</i>	endpoint number

Function Class:

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

10.53.4.31 void usb_lld_start_out (**USBDriver** * *usbp*, **usbep_t** *ep*)

Starts a receive operation on an OUT endpoint.

Parameters

in	<i>usbp</i>	pointer to the USBDriver object
in	<i>ep</i>	endpoint number

Function Class:

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

10.53.4.32 void usb_lld_start_in ([USBDriver](#) * *usbp*, [usbep_t](#) *ep*)

Starts a transmit operation on an IN endpoint.

Parameters

in	<i>usbp</i>	pointer to the USBDriver object
in	<i>ep</i>	endpoint number

Function Class:

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

10.53.4.33 void usb_lld_stall_out ([USBDriver](#) * *usbp*, [usbep_t](#) *ep*)

Brings an OUT endpoint in the stalled state.

Parameters

in	<i>usbp</i>	pointer to the USBDriver object
in	<i>ep</i>	endpoint number

Function Class:

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

10.53.4.34 void usb_lld_stall_in ([USBDriver](#) * *usbp*, [usbep_t](#) *ep*)

Brings an IN endpoint in the stalled state.

Parameters

in	<i>usbp</i>	pointer to the USBDriver object
in	<i>ep</i>	endpoint number

Function Class:

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

10.53.4.35 void usb_lld_clear_out ([USBDriver](#) * *usbp*, [usbep_t](#) *ep*)

Brings an OUT endpoint in the active state.

Parameters

in	<i>usbp</i>	pointer to the USBDriver object
in	<i>ep</i>	endpoint number

Function Class:

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

10.53.4.36 void usb_lld_clear_in(*USBDriver* * *usbp*, *usbep_t* *ep*)

Brings an IN endpoint in the active state.

Parameters

in	<i>usbp</i>	pointer to the <i>USBDriver</i> object
in	<i>ep</i>	endpoint number

Function Class:

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

10.53.5 Variable Documentation

10.53.5.1 *USBDriver USBD1*

USB1 driver identifier.

10.53.5.2 *USBInEndpointState* { ... } in

IN EP0 state.

10.53.5.3 *USBOutEndpointState* { ... } out

OUT EP0 state.

10.53.6 Define Documentation

10.53.6.1 #define *USB_DESC_INDEX*(*i*) ((*uint8_t*)(*i*))

Helper macro for index values into descriptor strings.

10.53.6.2 #define *USB_DESC_BYTE*(*b*) ((*uint8_t*)(*b*))

Helper macro for byte values into descriptor strings.

10.53.6.3 #define *USB_DESC_WORD*(*w*)

Value:

```
(uint8_t) ((w) & 255), \  
(uint8_t) (((w) >> 8) & 255)
```

Helper macro for word values into descriptor strings.

10.53.6.4 #define *USB_DESC_BCD*(*bcd*)

Value:

```
(uint8_t) ((bcd) & 255),
(uint8_t) (((bcd) >> 8) & 255)
```

Helper macro for BCD values into descriptor strings.

**10.53.6.5 #define USB_DESC_DEVICE(*bcdUSB*, *bDeviceClass*, *bDeviceSubClass*, *bDeviceProtocol*, *bMaxPacketSize*,
idVendor, *idProduct*, *bcdDevice*, *iManufacturer*, *iProduct*, *iSerialNumber*, *bNumConfigurations*)**

Value:

```
USB_DESC_BYTE (18),
USB_DESC_BYTE (USB_DESCRIPTOR_DEVICE),
USB_DESC_BCD (bcdUSB),
USB_DESC_BYTE (bDeviceClass),
USB_DESC_BYTE (bDeviceSubClass),
USB_DESC_BYTE (bDeviceProtocol),
USB_DESC_BYTE (bMaxPacketSize),
USB_DESC_WORD (idVendor),
USB_DESC_WORD (idProduct),
USB_DESC_BCD (bcdDevice),
USB_DESC_INDEX (iManufacturer),
USB_DESC_INDEX (iProduct),
USB_DESC_INDEX (iSerialNumber),
USB_DESC_BYTE (bNumConfigurations)
```

Device Descriptor helper macro.

**10.53.6.6 #define USB_DESC_CONFIGURATION(*wTotalLength*, *bNumInterfaces*, *bConfigurationValue*, *iConfiguration*,
bmAttributes, *bMaxPower*)**

Value:

```
USB_DESC_BYTE (9),
USB_DESC_BYTE (USB_DESCRIPTOR_CONFIGURATION),
USB_DESC_WORD (wTotalLength),
USB_DESC_BYTE (bNumInterfaces),
USB_DESC_BYTE (bConfigurationValue),
USB_DESC_INDEX (iConfiguration),
USB_DESC_BYTE (bmAttributes),
USB_DESC_BYTE (bMaxPower)
```

Configuration Descriptor helper macro.

**10.53.6.7 #define USB_DESC_INTERFACE(*bInterfaceNumber*, *bAlternateSetting*, *bNumEndpoints*, *bInterfaceClass*,
bInterfaceSubClass, *bInterfaceProtocol*, *iInterface*)**

Value:

```
USB_DESC_BYTE (9),
USB_DESC_BYTE (USB_DESCRIPTOR_INTERFACE),
USB_DESC_BYTE (bInterfaceNumber),
USB_DESC_BYTE (bAlternateSetting),
USB_DESC_BYTE (bNumEndpoints),
USB_DESC_BYTE (bInterfaceClass),
USB_DESC_BYTE (bInterfaceSubClass),
USB_DESC_BYTE (bInterfaceProtocol),
USB_DESC_INDEX (iInterface)
```

Interface Descriptor helper macro.

**10.53.6.8 #define USB_DESC_INTERFACE_ASSOCIATION(*bFirstInterface*, *bInterfaceCount*, *bFunctionClass*,
bFunctionSubClass, *bFunctionProtocol*, *iInterface*)**

Value:

```
USB_DESC_BYT(8),  
USB_DESC_BYT(USB_DESCRIPTOR_INTERFACE_ASSOCIATION),  
USB_DESC_BYT(bFirstInterface),  
USB_DESC_BYT(bInterfaceCount),  
USB_DESC_BYT(bFunctionClass),  
USB_DESC_BYT(bFunctionSubClass),  
USB_DESC_BYT(bFunctionProtocol),  
USB_DESC_INDX(iInterface)
```

Interface Association Descriptor helper macro.

10.53.6.9 #define USB_DESC_ENDPOINT(*bEndpointAddress*, *bmAttributes*, *wMaxPacketSize*, *bInterval*)

Value:

```
USB_DESC_BYT(7),  
USB_DESC_BYT(USB_DESCRIPTOR_ENDPOINT),  
USB_DESC_BYT(bEndpointAddress),  
USB_DESC_BYT(bmAttributes),  
USB_DESC_WD(wMaxPacketSize),  
USB_DESC_BYT(bInterval)
```

Endpoint Descriptor helper macro.

10.53.6.10 #define USB_EP_MODE_TYPE 0x0003

Endpoint type mask.

10.53.6.11 #define USB_EP_MODE_TYPE_CTRL 0x0000

Control endpoint.

10.53.6.12 #define USB_EP_MODE_TYPE_ISOC 0x0001

Isochronous endpoint.

10.53.6.13 #define USB_EP_MODE_TYPE_BULK 0x0002

Bulk endpoint.

10.53.6.14 #define USB_EP_MODE_TYPE_INTR 0x0003

Interrupt endpoint.

10.53.6.15 #define USB_EP_MODE_LINEAR_BUFFER 0x0000

Linear buffer mode.

10.53.6.16 #define USB_EP_MODE_QUEUE_BUFFER 0x0010

Queue buffer mode.

```
10.53.6.17 #define usbGetDriverState( usbp ) ((usbp)->state)
```

Returns the driver state.

Parameters

in *usbp* pointer to the [USBDriver](#) object

Returns

The driver state.

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.

```
10.53.6.18 #define usbFetchWord( p ) ((uint16_t)*(p) | ((uint16_t)*((p) + 1) << 8))
```

Fetches a 16 bits word value from an USB message.

Parameters

in *p* pointer to the 16 bits word

Function Class:

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

```
10.53.6.19 #define usbConnectBus( usbp ) usb_lld_connect_bus(usbp)
```

Connects the USB device.

Parameters

in *usbp* pointer to the [USBDriver](#) object

Function Class:

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

```
10.53.6.20 #define usbDisconnectBus( usbp ) usb_lld_disconnect_bus(usbp)
```

Disconnect the USB device.

Parameters

in *usbp* pointer to the [USBDriver](#) object

Function Class:

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

```
10.53.6.21 #define usbGetFrameNumber( usbp ) usb_lld_get_frame_number(usbp)
```

Returns the current frame number.

Parameters

in *usbp* pointer to the `USBDriver` object

Returns

The current frame number.

Function Class:

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

```
10.53.6.22 #define usbGetTransmitStatusI( usbp, ep ) ((usbp)>transmitting & (1 << (ep)))
```

Returns the status of an IN endpoint.

Parameters

in *usbp* pointer to the `USBDriver` object

in *ep* endpoint number

Returns

The operation status.

Return values

FALSE Endpoint ready.

TRUE Endpoint transmitting.

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.

```
10.53.6.23 #define usbGetReceiveStatusI( usbp, ep ) ((usbp)>receiving & (1 << (ep)))
```

Returns the status of an OUT endpoint.

Parameters

in *usbp* pointer to the `USBDriver` object

in *ep* endpoint number

Returns

The operation status.

Return values

FALSE Endpoint ready.

TRUE Endpoint receiving.

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.

```
10.53.6.24 #define usbGetReceiveTransactionSizeI( usbp, ep ) usb_lld_get_transaction_size(usbp, ep)
```

Returns the exact size of a receive transaction.

The received size can be different from the size specified in `usbStartReceiveI()` because the last packet could have a size different from the expected one.

Parameters

in	<code>usbp</code>	pointer to the <code>USBDriver</code> object
in	<code>ep</code>	endpoint number

Returns

Received data size.

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.

10.53.6.25 #define `usbSetupTransfer(usbp, buf, n, endcb)`

Value:

```
{
    (usbp)->ep0next  = (buf);
    (usbp)->ep0n      = (n);
    (usbp)->ep0endcb = (endcb);
}
```

Request transfer setup.

This macro is used by the request handling callbacks in order to prepare a transaction over the endpoint zero.

Parameters

in	<code>usbp</code>	pointer to the <code>USBDriver</code> object
in	<code>buf</code>	pointer to a buffer for the transaction data
in	<code>n</code>	number of bytes to be transferred
in	<code>endcb</code>	callback to be invoked after the transfer or NULL

Function Class:

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

10.53.6.26 #define `usbReadSetup(usbp, ep, buf) usb_lld_read_setup(usbp, ep, buf)`

Reads a setup packet from the dedicated packet buffer.

This function must be invoked in the context of the `setup_cb` callback in order to read the received setup packet.

Precondition

In order to use this function the endpoint must have been initialized as a control endpoint.

Note

This function can be invoked both in thread and IRQ context.

Parameters

in	<code>usbp</code>	pointer to the <code>USBDriver</code> object
in	<code>ep</code>	endpoint number
out	<code>buf</code>	buffer where to copy the packet data

Function Class:

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

10.53.6.27 #define _usb_isr_invoke_event_cb(*usbp*, *evt*)

Value:

```
{
    if (((usbp) ->config->event_cb) != NULL)
        (usbp) ->config->event_cb(usbp, evt);
}
```

Common ISR code, usb event callback.

Parameters

in	<i>usbp</i>	pointer to the USBDriver object
in	<i>evt</i>	USB event code

Function Class:

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

10.53.6.28 #define _usb_isr_invoke_sof_cb(*usbp*)

Value:

```
{
    if (((usbp) ->config->sof_cb) != NULL)
        (usbp) ->config->sof_cb(usbp);
}
```

Common ISR code, SOF callback.

Parameters

in	<i>usbp</i>	pointer to the USBDriver object
----	-------------	---

Function Class:

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

10.53.6.29 #define _usb_isr_invoke_setup_cb(*usbp*, *ep*)

Value:

```
{
    (usbp) ->epc[ep] ->setup_cb(usbp, ep);
}
```

Common ISR code, setup packet callback.

Parameters

in	<i>usbp</i>	pointer to the USBDriver object
in	<i>ep</i>	endpoint number

Function Class:

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

10.53.6.30 #define _usb_isr_invoke_in_cb(usbp, ep)

Value:

```
{
    \
    (usbp)->transmitting &= ~(1 << (ep));
    \
    (usbp)->epc[ep]->in_cb(usbp, ep);
}
```

Common ISR code, IN endpoint callback.

Parameters

in	<i>usbp</i>	pointer to the USBDriver object
in	<i>ep</i>	endpoint number

Function Class:

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

10.53.6.31 #define _usb_isr_invoke_out_cb(usbp, ep)

Value:

```
{
    \
    (usbp)->receiving &= ~(1 << (ep));
    \
    (usbp)->epc[ep]->out_cb(usbp, ep);
}
```

Common ISR code, OUT endpoint event.

Parameters

in	<i>usbp</i>	pointer to the USBDriver object
in	<i>ep</i>	endpoint number

Function Class:

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

10.53.6.32 #define USB_MAX_ENDPOINTS 4

Maximum endpoint address.

10.53.6.33 #define USB_EP0_STATUS_STAGE USB_EP0_STATUS_STAGE_SW

Status stage handling method.

10.53.6.34 #define USB_SET_ADDRESS_MODE USB_EARLY_SET_ADDRESS

The address can be changed immediately upon packet reception.

```
10.53.6.35 #define USB_SET_ADDRESS_ACK_HANDLING USB_SET_ADDRESS_ACK_SW
```

Method for set address acknowledge.

```
10.53.6.36 #define PLATFORM_USB_USE_USB1 FALSE
```

USB driver enable switch.

If set to TRUE the support for USB1 is included.

```
10.53.6.37 #define usb_lld_get_transaction_size( usbp, ep ) ((usbp)->epc[ep]->out_state->rxcnt)
```

Returns the exact size of a receive transaction.

The received size can be different from the size specified in [usbStartReceiveI\(\)](#) because the last packet could have a size different from the expected one.

Precondition

The OUT endpoint must have been configured in transaction mode in order to use this function.

Parameters

in	usbp	pointer to the USBDriver object
in	ep	endpoint number

Returns

Received data size.

Function Class:

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

```
10.53.6.38 #define usb_lld_connect_bus( usbp )
```

Connects the USB device.

Function Class:

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

```
10.53.6.39 #define usb_lld_disconnect_bus( usbp )
```

Disconnect the USB device.

Function Class:

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

10.53.7 Typedef Documentation

```
10.53.7.1 typedef struct USBDriver USBDriver
```

Type of a structure representing an USB driver.

10.53.7.2 `typedef uint8_t usbep_t`

Type of an endpoint identifier.

10.53.7.3 `typedef void(* usbcallback_t)(USBDriver *usbp)`

Type of an USB generic notification callback.

Parameters

in *usbp* pointer to the `USBDriver` object triggering the callback

10.53.7.4 `typedef void(* usbepcallback_t)(USBDriver *usbp, usbep_t ep)`

Type of an USB endpoint callback.

Parameters

in *usbp* pointer to the `USBDriver` object triggering the callback

in *ep* endpoint number

10.53.7.5 `typedef void(* usbeventcb_t)(USBDriver *usbp, usbevent_t event)`

Type of an USB event notification callback.

Parameters

in *usbp* pointer to the `USBDriver` object triggering the callback

in *event* event type

10.53.7.6 `typedef bool_t(* usbreqhandler_t)(USBDriver *usbp)`

Type of a requests handler callback.

The request is encoded in the `usb_setup` buffer.

Parameters

in *usbp* pointer to the `USBDriver` object triggering the callback

Returns

The request handling exit code.

Return values

FALSE Request not recognized by the handler.

TRUE Request handled.

10.53.7.7 `typedef const USBDescriptor*(* usbgetdescriptor_t)(USBDriver *usbp, uint8_t dtype, uint8_t dindex, uint16_t lang)`

Type of an USB descriptor-retrieving callback.

10.53.8 Enumeration Type Documentation

10.53.8.1 enum usbstate_t

Type of a driver state machine possible states.

Enumerator:

USB_UNINIT Not initialized.

USB_STOP Stopped.

USB_READY Ready, after bus reset.

USB_SELECTED Address assigned.

USB_ACTIVE Active, configuration selected.

10.53.8.2 enum usbepstatus_t

Type of an endpoint status.

Enumerator:

EP_STATUS_DISABLED Endpoint not active.

EP_STATUS_STALLED Endpoint opened but stalled.

EP_STATUS_ACTIVE Active endpoint.

10.53.8.3 enum usbep0state_t

Type of an endpoint zero state machine states.

Enumerator:

USB_EP0_WAITING_SETUP Waiting for SETUP data.

USB_EP0_TX Transmitting.

USB_EP0_WAITING_TX0 Waiting transmit 0.

USB_EP0_WAITING_STS Waiting status.

USB_EP0_RX Receiving.

USB_EP0_SENDING_STS Sending status.

USB_EP0_ERROR Error, EP0 stalled.

10.53.8.4 enum usbevent_t

Type of an enumeration of the possible USB events.

Enumerator:

USB_EVENT_RESET Driver has been reset by host.

USB_EVENT_ADDRESS Address assigned.

USB_EVENT_CONFIGURED Configuration selected.

USB_EVENT_SUSPEND Entering suspend mode.

USB_EVENT_WAKEUP Leaving suspend mode.

USB_EVENT_STALLED Endpoint 0 error, stalled.

10.54 HAL

10.54.1 Detailed Description

Hardware Abstraction Layer. Under ChibiOS/RT the set of the various device driver interfaces is called the HAL subsystem: Hardware Abstraction Layer. The HAL is the abstract interface between ChibiOS/RT application and hardware.

10.54.2 HAL Device Drivers Architecture

A device driver is usually split in two layers:

- High Level Device Driver (**HLD**). This layer contains the definitions of the driver's APIs and the platform independent part of the driver.

An HLD is composed by two files:

- `<driver>.c`, the HLD implementation file. This file must be included in the Makefile in order to use the driver.
- `<driver>.h`, the HLD header file. This file is implicitly included by the HAL header file `hal.h`.

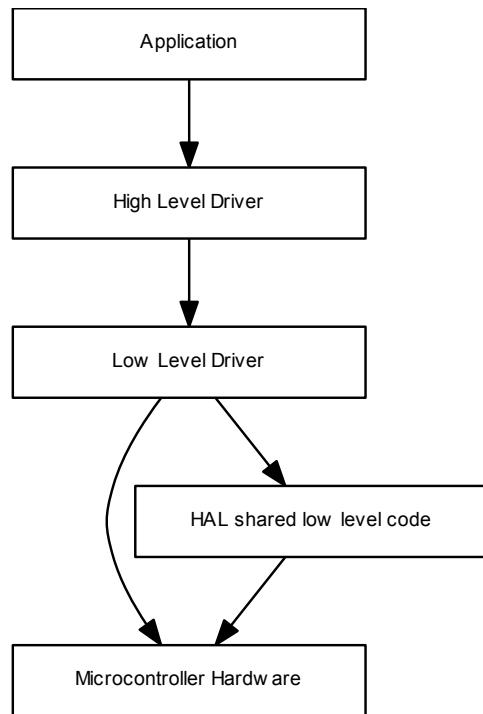
- Low Level Device Driver (**LLD**). This layer contains the platform dependent part of the driver.

A LLD is composed by two files:

- `<driver>_lld.c`, the LLD implementation file. This file must be included in the Makefile in order to use the driver.
- `<driver>_lld.h`, the LLD header file. This file is implicitly included by the HLD header file.

The LLD may be not present in those drivers that do not access the hardware directly but through other device drivers, as example the MMC_SPI driver uses the SPI and PAL drivers in order to implement its functionalities.

10.54.2.1 Diagram



Modules

- [ADC Driver](#)
Generic ADC Driver.
- [CAN Driver](#)
Generic CAN Driver.
- [EXT Driver](#)
Generic EXT Driver.
- [GPT Driver](#)
Generic GPT Driver.
- [HAL Driver](#)
Hardware Abstraction Layer.
- [I2C Driver](#)
Generic I2C Driver.
- [I2S Driver](#)
Generic I2S Driver.
- [ICU Driver](#)
Generic ICU Driver.
- [Abstract I/O Block Device](#)
- [Abstract I/O Channel](#)
- [MAC Driver](#)
Generic MAC driver.
- [MMC over SPI Driver](#)

- *Generic MMC driver.*
- [MMC/SD Block Device](#)
- [PAL Driver](#)
 - I/O Ports Abstraction Layer.*
 - [PWM Driver](#)
 - Generic PWM Driver.*
 - [RTC Driver](#)
 - Real Time Clock Abstraction Layer.*
 - [SDC Driver](#)
 - Generic SD Card Driver.*
 - [Serial Driver](#)
 - Generic Serial Driver.*
 - [Serial over USB Driver](#)
 - Serial over USB Driver.*
 - [SPI Driver](#)
 - Generic SPI Driver.*
 - [Time Measurement Driver](#)
 - Time Measurement unit.*
 - [UART Driver](#)
 - Generic UART Driver.*
 - [USB Driver](#)
 - Generic USB Driver.*
 - [Configuration](#)
 - HAL Configuration.*

10.55 Configuration

10.55.1 Detailed Description

HAL Configuration. The file `halconf.h` contains the high level settings for all the drivers supported by the HAL. The low level, platform dependent, settings are contained in the `mcuconf.h` file instead and are described in the various platforms reference manuals.

Drivers enable switches

- `#define HAL_USE_TM TRUE`
Enables the TM subsystem.
- `#define HAL_USE_PAL TRUE`
Enables the PAL subsystem.
- `#define HAL_USE_ADC TRUE`
Enables the ADC subsystem.
- `#define HAL_USE_CAN TRUE`
Enables the CAN subsystem.
- `#define HAL_USE_EXT TRUE`
Enables the EXT subsystem.
- `#define HAL_USE_GPT TRUE`
Enables the GPT subsystem.
- `#define HAL_USE_I2C TRUE`
Enables the I2C subsystem.
- `#define HAL_USE_ICU TRUE`

- `#define HAL_USE_MAC TRUE`
Enables the MAC subsystem.
- `#define HAL_USE_MMC_SPI TRUE`
Enables the MMC_SPI subsystem.
- `#define HAL_USE_PWM TRUE`
Enables the PWM subsystem.
- `#define HAL_USE_RTC FALSE`
Enables the RTC subsystem.
- `#define HAL_USE_SDC TRUE`
Enables the SDC subsystem.
- `#define HAL_USE_SERIAL TRUE`
Enables the SERIAL subsystem.
- `#define HAL_USE_SERIAL_USB TRUE`
Enables the SERIAL over USB subsystem.
- `#define HAL_USE_SPI TRUE`
Enables the SPI subsystem.
- `#define HAL_USE_UART TRUE`
Enables the UART subsystem.
- `#define HAL_USE_USB TRUE`
Enables the USB subsystem.

ADC driver related setting

- `#define ADC_USE_WAIT TRUE`
Enables synchronous APIs.
- `#define ADC_USE_MUTUAL_EXCLUSION TRUE`
Enables the `adcAcquireBus()` and `adcReleaseBus()` APIs.

CAN driver related setting

- `#define CAN_USE_SLEEP_MODE TRUE`
Sleep mode related APIs inclusion switch.

I2C driver related setting

- `#define I2C_USE_MUTUAL_EXCLUSION TRUE`
Enables the mutual exclusion APIs on the I2C bus.

MAC driver related setting

- `#define MAC_USE_ZERO_COPY TRUE`
Enables an event sources for incoming packets.
- `#define MAC_USE_EVENTS TRUE`
Enables an event sources for incoming packets.

MMC_SPI driver related setting

- `#define MMC_NICE_WAITING TRUE`
Delays insertions.

SDC driver related setting

- `#define SDC_INIT_RETRY 100`
Number of initialization attempts before rejecting the card.
- `#define SDC_MMC_SUPPORT TRUE`
Include support for MMC cards.
- `#define SDC_NICE_WAITING TRUE`
Delays insertions.

SERIAL driver related setting

- `#define SERIAL_DEFAULT_BITRATE 38400`
Default bit rate.
- `#define SERIAL_BUFFERS_SIZE 16`
Serial buffers size.

SERIAL_USB driver related setting

- `#define SERIAL_USB_BUFFERS_SIZE 64`
Serial over USB buffers size.

SPI driver related setting

- `#define SPI_USE_WAIT TRUE`
Enables synchronous APIs.
- `#define SPI_USE_MUTUAL_EXCLUSION TRUE`
Enables the `spiAcquireBus()` and `spiReleaseBus()` APIs.

10.55.2 Define Documentation

10.55.2.1 `#define HAL_USE_TM TRUE`

Enables the TM subsystem.

10.55.2.2 `#define HAL_USE_PAL TRUE`

Enables the PAL subsystem.

10.55.2.3 `#define HAL_USE_ADC TRUE`

Enables the ADC subsystem.

10.55.2.4 `#define HAL_USE_CAN TRUE`

Enables the CAN subsystem.

10.55.2.5 `#define HAL_USE_EXT TRUE`

Enables the EXT subsystem.

10.55.2.6 #define HAL_USE_GPT TRUE

Enables the GPT subsystem.

10.55.2.7 #define HAL_USE_I2C TRUE

Enables the I2C subsystem.

10.55.2.8 #define HAL_USE_ICU TRUE

Enables the ICU subsystem.

10.55.2.9 #define HAL_USE_MAC TRUE

Enables the MAC subsystem.

10.55.2.10 #define HAL_USE_MMC_SPI TRUE

Enables the MMC_SPI subsystem.

10.55.2.11 #define HAL_USE_PWM TRUE

Enables the PWM subsystem.

10.55.2.12 #define HAL_USE_RTC FALSE

Enables the RTC subsystem.

10.55.2.13 #define HAL_USE_SDC TRUE

Enables the SDC subsystem.

10.55.2.14 #define HAL_USE_SERIAL TRUE

Enables the SERIAL subsystem.

10.55.2.15 #define HAL_USE_SERIAL_USB TRUE

Enables the SERIAL over USB subsystem.

10.55.2.16 #define HAL_USE_SPI TRUE

Enables the SPI subsystem.

10.55.2.17 #define HAL_USE_UART TRUE

Enables the UART subsystem.

10.55.2.18 #define HAL_USE_USB TRUE

Enables the USB subsystem.

10.55.2.19 #define ADC_USE_WAIT TRUE

Enables synchronous APIs.

Note

Disabling this option saves both code and data space.

10.55.2.20 #define ADC_USE_MUTUAL_EXCLUSION TRUE

Enables the `adcAcquireBus()` and `adcReleaseBus()` APIs.

Note

Disabling this option saves both code and data space.

10.55.2.21 #define CAN_USE_SLEEP_MODE TRUE

Sleep mode related APIs inclusion switch.

10.55.2.22 #define I2C_USE_MUTUAL_EXCLUSION TRUE

Enables the mutual exclusion APIs on the I2C bus.

10.55.2.23 #define MAC_USE_ZERO_COPY TRUE

Enables an event sources for incoming packets.

10.55.2.24 #define MAC_USE_EVENTS TRUE

Enables an event sources for incoming packets.

10.55.2.25 #define MMC_NICE_WAITING TRUE

Delays insertions.

If enabled this options inserts delays into the MMC waiting routines releasing some extra CPU time for the threads with lower priority, this may slow down the driver a bit however. This option is recommended also if the SPI driver does not use a DMA channel and heavily loads the CPU.

10.55.2.26 #define SDC_INIT_RETRY 100

Number of initialization attempts before rejecting the card.

Note

Attempts are performed at 10mS intervals.

10.55.2.27 #define SDC_MMC_SUPPORT TRUE

Include support for MMC cards.

Note

MMC support is not yet implemented so this option must be kept at FALSE.

10.55.2.28 #define SDC_NICE_WAITING TRUE

Delays insertions.

If enabled this options inserts delays into the MMC waiting routines releasing some extra CPU time for the threads with lower priority, this may slow down the driver a bit however.

10.55.2.29 #define SERIAL_DEFAULT_BITRATE 38400

Default bit rate.

Configuration parameter, this is the baud rate selected for the default configuration.

10.55.2.30 #define SERIAL_BUFFERS_SIZE 16

Serial buffers size.

Configuration parameter, you can change the depth of the queue buffers depending on the requirements of your application.

Note

The default is 64 bytes for both the transmission and receive buffers.

10.55.2.31 #define SERIAL_USB_BUFFERS_SIZE 64

Serial over USB buffers size.

Configuration parameter, the buffer size must be a multiple of the USB data endpoint maximum packet size.

Note

The default is 64 bytes for both the transmission and receive buffers.

10.55.2.32 #define SPI_USE_WAIT TRUE

Enables synchronous APIs.

Note

Disabling this option saves both code and data space.

10.55.2.33 #define SPI_USE_MUTUAL_EXCLUSION TRUE

Enables the `spiAcquireBus()` and `spiReleaseBus()` APIs.

Note

Disabling this option saves both code and data space.

10.56 Various

10.56.1 Detailed Description

Utilities Library. This is a collection of useful library code that is not part of the base kernel services.

Notes

The library code does not follow the same naming convention of the system APIs in order to make very clear that it is not "core" code.

The main difference is that library code is not formally tested in the test suite but through usage in the various demo applications.

Modules

- [C++ Wrapper](#)

C++ wrapper module.

- [Memory Streams](#)

Memory Streams.

- [Periodic Events Timer](#)

Periodic Event Timer.

- [Command Shell](#)

Small extendible command line shell.

- [RTC time conversion utilities](#)

RTC time conversion utilities.

- [System formatted print](#)

System formatted print service.

10.57 C++ Wrapper

10.57.1 Detailed Description

C++ wrapper module. This module allows to use the ChibiOS/RT functionalities from C++ as classes and objects rather the traditional "C" APIs.

Namespaces

- namespace [chibios_rt](#)

ChibiOS kernel-related classes and interfaces.

10.58 Memory Streams

10.58.1 Detailed Description

Memory Streams. This module allows to use a memory area (RAM or ROM) using a [Abstract Sequential Streams](#) interface.

Data Structures

- struct [MemStreamVMT](#)
MemStream virtual methods table.
- struct [MemoryStream](#)
Memory stream object.

Functions

- void [msObjectInit](#) ([MemoryStream](#) *msp, uint8_t *buffer, size_t size, size_t eos)
Memory stream object initialization.

Defines

- #define [_memory_stream_data](#)
RamStream specific data.

10.58.2 Function Documentation

10.58.2.1 void msObjectInit ([MemoryStream](#) * *msp*, [uint8_t](#) * *buffer*, [size_t](#) *size*, [size_t](#) *eos*)

Memory stream object initialization.

Parameters

out	<i>msp</i>	pointer to the MemoryStream object to be initialized
in	<i>buffer</i>	pointer to the memory buffer for the memory stream
in	<i>size</i>	total size of the memory stream buffer
in	<i>eos</i>	initial End Of Stream offset. Normally you need to put this to zero for RAM buffers or equal to <i>size</i> for ROM streams.

10.58.3 Define Documentation

10.58.3.1 #define [_memory_stream_data](#)

Value:

```
_base_sequential_stream_data
/* Pointer to the stream buffer.*/
uint8\_t           *buffer;
/* Size of the stream.*/
size\_t            size;
/* Current end of stream.*/
size\_t            eos;
/* Current read offset.*/
size\_t            offset;
```

RamStream specific data.

10.59 Periodic Events Timer

10.59.1 Detailed Description

Periodic Event Timer. This timer generates an event at regular intervals. The listening threads can use the event to perform time related activities. Multiple threads can listen to the same timer.

Data Structures

- struct [EvTimer](#)
Event timer structure.

Functions

- void [evtStart](#) ([EvTimer](#) **etp*)
Starts the timer.
- void [evtStop](#) ([EvTimer](#) **etp*)
Stops the timer.

Defines

- #define [evtInit](#)(*etp*, *time*)
Initializes an [EvTimer](#) structure.

10.59.2 Function Documentation

10.59.2.1 void evtStart ([EvTimer](#) * *etp*)

Starts the timer.

If the timer was already running then the function has no effect.

Parameters

etp pointer to an initialized [EvTimer](#) structure.

Here is the call graph for this function:



10.59.2.2 void evtStop ([EvTimer](#) * *etp*)

Stops the timer.

If the timer was already stopped then the function has no effect.

Parameters

etp pointer to an initialized [EvTimer](#) structure.

10.59.3 Define Documentation

10.59.3.1 #define evtInit(etp, time)

Value:

```
{
    chEvtInit (& (etp) ->et_es);
    (etp) ->et_vt.vt_func = NULL;
    (etp) ->et_interval = (time);
}
```

Initializes an [EvTimer](#) structure.

Parameters

<i>etp</i>	the EvTimer structure to be initialized
<i>time</i>	the interval in system ticks

10.60 Command Shell

10.60.1 Detailed Description

Small extendible command line shell. This module implements a generic extendible command line interface. The CLI just requires an I/O channel ([BaseChannel](#)), more commands can be added to the shell using the configuration structure.

Data Structures

- struct [ShellCommand](#)
Custom command entry type.
- struct [ShellConfig](#)
Shell descriptor type.

Functions

- void [shellInit](#) (void)
Shell manager initialization.
- void [shellExit](#) (msg_t msg)
Terminates the shell.
- Thread * [shellCreate](#) (const ShellConfig *scp, size_t size, t prio_t prio)
Spawns a new shell.
- Thread * [shellCreateStatic](#) (const ShellConfig *scp, void *wsp, size_t size, t prio_t prio)
Create statically allocated shell thread.
- bool_t [shellGetLine](#) (BaseSequentialStream *chp, char *line, unsigned size)
Reads a whole line from the input channel.

Variables

- EventSource [shell_terminated](#)
Shell termination event source.

Defines

- #define **SHELL_MAX_LINE_LENGTH** 64
Shell maximum input line length.
- #define **SHELL_MAX_ARGUMENTS** 4
Shell maximum arguments per command.

TypeDefs

- typedef void(* **shellcmd_t**)(BaseSequentialStream *chp, int argc, char *argv[])
Command handler function type.

10.60.2 Function Documentation

10.60.2.1 void shellInit(void)

Shell manager initialization.

Function Class:

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

10.60.2.2 void shellExit(msg_t msg)

Terminates the shell.

Note

Must be invoked from the command handlers.
Does not return.

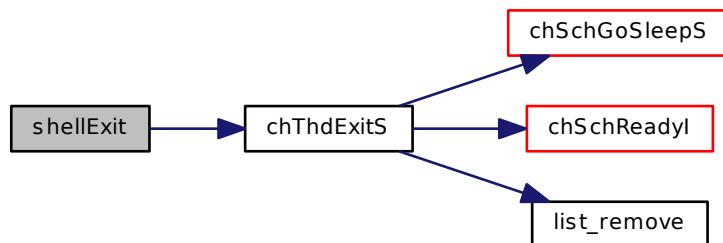
Parameters

in msg shell 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:



10.60.2.3 Thread * shellCreate (const ShellConfig * scp, size_t size, tprio_t prio)

Spawns a new shell.

Precondition

CH_USE_HEAP and CH_USE_DYNAMIC must be enabled.

Parameters

in	scp	pointer to a <code>ShellConfig</code> object
in	size	size of the shell working area to be allocated
in	prio	priority level for the new shell

Returns

A pointer to the shell thread.

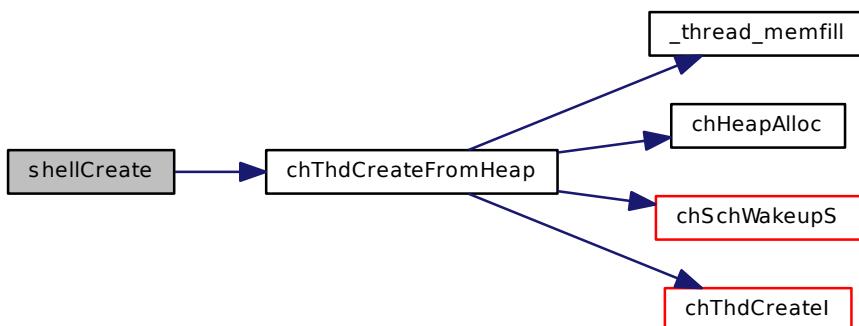
Return values

`NULL` thread creation failed because memory allocation.

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:



10.60.2.4 Thread * shellCreateStatic (const ShellConfig * scp, void * wsp, size_t size, tprio_t prio)

Create statically allocated shell thread.

Parameters

in	scp	pointer to a <code>ShellConfig</code> object
in	wsp	pointer to a working area dedicated to the shell thread stack
in	size	size of the shell working area
in	prio	priority level for the new shell

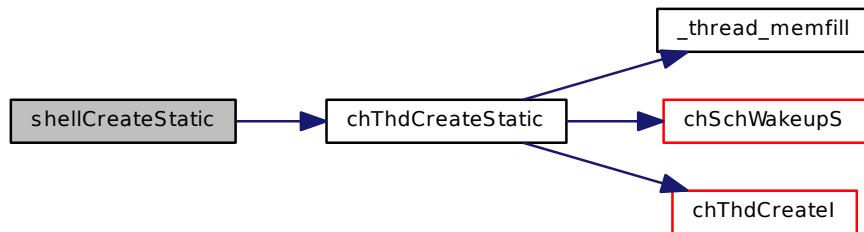
Returns

A pointer to the shell 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:



10.60.2.5 `bool_t shellGetLine(BaseSequentialStream * chp, char * line, unsigned size)`

Reads a whole line from the input channel.

Parameters

in	<i>chp</i>	pointer to a <code>BaseSequentialStream</code> object
in	<i>line</i>	pointer to the line buffer
in	<i>size</i>	buffer maximum length

Returns

The operation status.

Return values

<i>TRUE</i>	the channel was reset or CTRL-D pressed.
<i>FALSE</i>	operation successful.

Function Class:

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

10.60.3 Variable Documentation

10.60.3.1 EventSource `shell_terminated`

Shell termination event source.

10.60.4 Define Documentation

10.60.4.1 `#define SHELL_MAX_LINE_LENGTH 64`

Shell maximum input line length.

10.60.4.2 #define SHELL_MAX_ARGUMENTS 4

Shell maximum arguments per command.

10.60.5 Typedef Documentation

10.60.5.1 `typedef void(* shellcmd_t)(BaseSequentialStream *chp, int argc, char *argv[])`

Command handler function type.

10.61 RTC time conversion utilities

10.61.1 Detailed Description

RTC time conversion utilities.

Functions

- `void rtcGetTimeTm (RTCDriver *rtcp, struct tm *timp)`
Gets raw time from RTC and converts it to canonicalized format.
- `void rtcSetTimeTm (RTCDriver *rtcp, struct tm *timp)`
Sets RTC time.
- `time_t rtcGetTimeUnixSec (RTCDriver *rtcp)`
Gets raw time from RTC and converts it to unix format.
- `void rtcSetTimeUnixSec (RTCDriver *rtcp, time_t tv_sec)`
Sets RTC time.
- `uint64_t rtcGetTimeUnixUsec (RTCDriver *rtcp)`
Gets raw time from RTC and converts it to unix format.
- `uint32_t rtcGetTimeFatFromCounter (RTCDriver *rtcp)`
Get current time in format suitable for usage in FatFS.
- `uint32_t rtcGetTimeFat (RTCDriver *rtcp)`
Get current time in format suitable for usage in FatFS.

10.61.2 Function Documentation

10.61.2.1 `void rtcGetTimeTm (RTCDriver * rtcp, struct tm * timp)`

Gets raw time from RTC and converts it to canonicalized format.

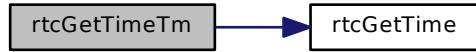
Parameters

in	<code>rtcp</code>	pointer to RTC driver structure
out	<code>timp</code>	pointer to a <code>tm</code> structure as defined in <code>time.h</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:



10.61.2.2 void rtcSetTimeTm (RTCDriver * *rtcp*, struct tm * *tmp*)

Sets RTC time.

Parameters

in	<i>rtcp</i>	pointer to RTC driver structure
out	<i>tmp</i>	pointer to a <code>tm</code> structure as defined in <code>time.h</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:



10.61.2.3 time_t rtcGetTimeUnixSec (RTCDriver * *rtcp*)

Gets raw time from RTC and converts it to unix format.

Parameters

in	<i>rtcp</i>	pointer to RTC driver structure
----	-------------	---------------------------------

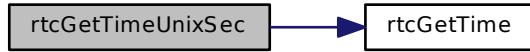
Returns

Unix time value in seconds.

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:



10.61.2.4 void rtcSetTimeUnixSec (RTCDriver * *rtcp*, time_t *tv_sec*)

Sets RTC time.

Parameters

in	<i>rtcp</i>	pointer to RTC driver structure
in	<i>tv_sec</i>	time specification

Returns

Unix time value in seconds.

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:



10.61.2.5 uint64_t rtcGetTimeUnixUsec (RTCDriver * *rtcp*)

Gets raw time from RTC and converts it to unix format.

Parameters

in	<i>rtcp</i>	pointer to RTC driver structure
----	-------------	---------------------------------

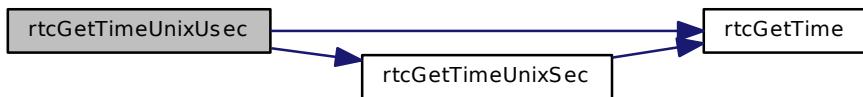
Returns

Unix time value in microseconds.

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:



10.61.2.6 uint32_t rtcGetTimeFatFromCounter (RTCDriver * *rtcp*)

Get current time in format suitable for usage in FatFS.

Parameters

in *rtcp* pointer to RTC driver structure

Returns

FAT time value.

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:



10.61.2.7 uint32_t rtcGetTimeFat (RTCDriver * *rtcp*)

Get current time in format suitable for usage in FatFS.

Parameters

in *rtcp* pointer to RTC driver structure

Returns

FAT time value.

Function Class:

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

10.62 System formatted print

10.62.1 Detailed Description

System formatted print service. This module implements printf()-like function able to send data to any module implementing a [BaseSequentialStream](#) interface.

Functions

- void [chvprintf](#) ([BaseSequentialStream](#) **chp*, const char **fmt*, va_list *ap*)
System formatted output function.
- int [chsnprintf](#) (char **str*, size_t *size*, const char **fmt*,...)
System formatted output function.

Defines

- #define [CHPRINTF_USE_FLOAT](#) FALSE
Float type support.

10.62.2 Function Documentation

10.62.2.1 void chvprintf ([BaseSequentialStream](#) * *chp*, const char * *fmt*, va_list *ap*)

System formatted output function.

This function implements a minimal vprintf() -like functionality with output on a [BaseSequentialStream](#). The general parameters format is: %[-][width|*][.precision|*][I|L]p. The following parameter types (p) are supported:

- **x** hexadecimal integer.
- **X** hexadecimal long.
- **o** octal integer.
- **O** octal long.
- **d** decimal signed integer.
- **D** decimal signed long.
- **u** decimal unsigned integer.
- **U** decimal unsigned long.
- **c** character.
- **s** string.

Parameters

in	<i>chp</i>	pointer to a BaseSequentialStream implementing object
in	<i>fmt</i>	formatting string
in	<i>ap</i>	list of parameters

Function Class:

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

10.62.2.2 int chsnprintf (char * str, size_t size, const char * fmt, ...)

System formatted output function.

This function implements a minimal `vprintf()`-like functionality with output on a [BaseSequentialStream](#). The general parameters format is: %[-][width|*][.precision|*][l|L]p. The following parameter types (p) are supported:

- **x** hexadecimal integer.
- **X** hexadecimal long.
- **o** octal integer.
- **O** octal long.
- **d** decimal signed integer.
- **D** decimal signed long.
- **u** decimal unsigned integer.
- **U** decimal unsigned long.
- **c** character.
- **s** string.

Parameters

in	<i>str</i>	pointer to a buffer
in	<i>size</i>	maximum size of the buffer
in	<i>fmt</i>	formatting string

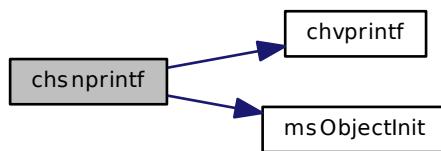
Returns

The size of the generated string.

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:



10.62.3 Define Documentation

10.62.3.1 #define CHPRINTF_USE_FLOAT FALSE

Float type support.

10.63 Test Runtime

10.63.1 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.

Data Structures

- struct `testcase`

Structure representing a test case.

Functions

- void `test_printfn` (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.
- void `test_cpu_pulse` (unsigned duration)
CPU pulse.
- `systime_t test_wait_tick` (void)
Delays execution until next system time tick.
- void `test_start_timer` (unsigned ms)
Starts the test timer.
- `msg_t TestThread` (void *p)
Test execution thread function.

Variables

- `bool_t test_timer_done`
Set to TRUE when the test timer reaches its deadline.

Defines

- `#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.

10.63.2 Function Documentation

10.63.2.1 void test_printn (uint32_t n)

Prints a decimal unsigned number.

Parameters

in *n* the number to be printed

10.63.2.2 void test_print (const char * msgp)

Prints a line without final end-of-line.

Parameters

in *msgp* the message

10.63.2.3 void test.println (const char * msgp)

Prints a line.

Parameters

in *msgp* the message

Here is the call graph for this function:



10.63.2.4 void test.emit_token (char token)

Emits a token into the tokens buffer.

Parameters

in *token* the token as a char

10.63.2.5 void test_terminate_threads (void)

Sets a termination request in all the test-spawned threads.

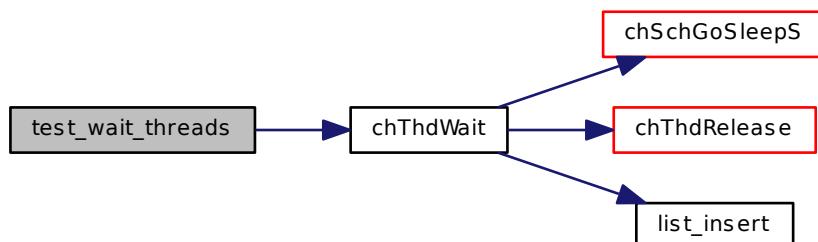
Here is the call graph for this function:



10.63.2.6 void test_wait_threads (void)

Waits for the completion of all the test-spawned threads.

Here is the call graph for this function:



10.63.2.7 void test_cpu_pulse (unsigned duration)

CPU pulse.

Note

The current implementation is not totally reliable.

Parameters

in duration CPU pulse duration in milliseconds

10.63.2.8 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:

**10.63.2.9 void test_start_timer(unsigned ms)**

Starts the test timer.

Parameters

in *ms* time in milliseconds

10.63.2.10 msg_t TestThread(void * p)

Test execution thread function.

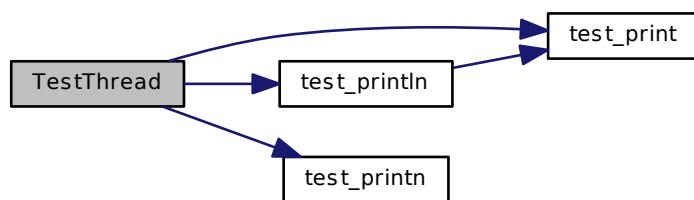
Parameters

in *p* pointer to a [BaseChannel](#) object for test output

Returns

A failure boolean value.

Here is the call graph for this function:

**10.63.3 Variable Documentation**

10.63.3.1 `bool_t test_timer_done`

Set to TRUE when the test timer reaches its deadline.

10.63.4 Define Documentation

10.63.4.1 `#define DELAY_BETWEEN_TESTS 200`

Delay inserted between test cases.

10.63.4.2 `#define TEST_NO_BENCHMARKS FALSE`

If TRUE then benchmarks are not included.

10.63.4.3 `#define test_fail(point)`

Value:

```
{
    _test_fail(point);
    return;
}
```

Test failure enforcement.

10.63.4.4 `#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

10.63.4.5 `#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

10.63.4.6 #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

10.63.4.7 #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)

10.64 External Components

ChibiOS/RT supports several external libraries through support interfaces and/or demos. Credit should be given to the original authors for making available such useful code.

The current list of supported component is:

- **uIP**, by Adam Dunkels at the Swedish Institute of Computer Science, [link](#).
- **IwIP**, many authors, [link](#).
- **FatFs**, by "ChaN", [link](#).

External components and libraries are not directly supported and are used "as is" or with minor integration patching.

Chapter 11

Namespace Documentation

11.1 chibios_rt Namespace Reference

11.1.1 Detailed Description

ChibiOS kernel-related classes and interfaces.

Data Structures

- class [System](#)
Class encapsulating the base system functionalities.
- class [Core](#)
Class encapsulating the base system functionalities.
- class [Timer](#)
Timer class.
- class [ThreadReference](#)
Thread reference class.
- class [BaseThread](#)
Abstract base class for a ChibiOS/RT thread.
- class [BaseStaticThread](#)
Static threads template class.
- class [CounterSemaphore](#)
Class encapsulating a semaphore.
- class [BinarySemaphore](#)
Class encapsulating a binary semaphore.
- class [Mutex](#)
Class encapsulating a mutex.
- class [CondVar](#)
Class encapsulating a conditional variable.
- class [EvtListener](#)
Class encapsulating an event listener.
- class [EvtSource](#)
Class encapsulating an event source.
- class [InQueue](#)
Class encapsulating an input queue.
- class [InQueueBuffer](#)
Template class encapsulating an input queue and its buffer.

- class [OutQueue](#)
Class encapsulating an output queue.
- class [OutQueueBuffer](#)
Template class encapsulating an output queue and its buffer.
- class [Mailbox](#)
Class encapsulating a mailbox.
- class [MailboxBuffer](#)
Template class encapsulating a mailbox and its messages buffer.
- class [MemoryPool](#)
Class encapsulating a memory pool.
- class [ObjectsPool](#)
Template class encapsulating a memory pool and its elements.
- class [BaseSequentialStreamInterface](#)
Interface of a [BaseSequentialStream](#).

Chapter 12

Data Structure Documentation

12.1 ADCConfig Struct Reference

12.1.1 Detailed Description

Driver configuration structure.

Note

It could be empty on some architectures.

```
#include <adc_lld.h>
```

12.2 ADCConversionGroup Struct Reference

12.2.1 Detailed Description

Conversion group configuration structure.

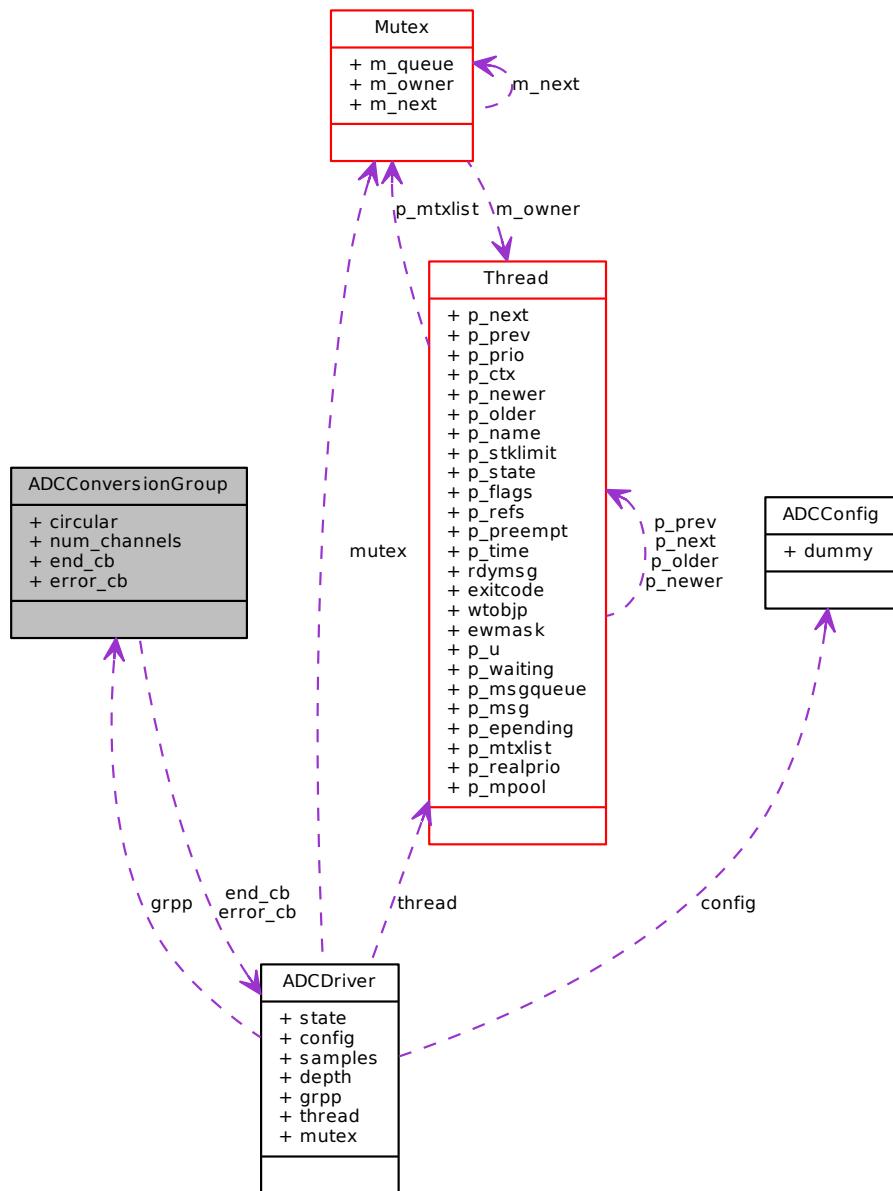
This implementation-dependent structure describes a conversion operation.

Note

Implementations may extend this structure to contain more, architecture dependent, fields.

```
#include <adc_lld.h>
```

Collaboration diagram for ADCConversionGroup:



Data Fields

- **bool_t circular**
Enables the circular buffer mode for the group.
- **adc_channels_num_t num_channels**
Number of the analog channels belonging to the conversion group.
- **adccallback_t end_cb**
Callback function associated to the group or NULL.
- **adccallback_t error_cb**
Error callback or NULL.

12.2.2 Field Documentation

12.2.2.1 `bool_t ADCConversionGroup::circular`

Enables the circular buffer mode for the group.

12.2.2.2 `adc_channels_num_t ADCConversionGroup::num_channels`

Number of the analog channels belonging to the conversion group.

12.2.2.3 `adccallback_t ADCConversionGroup::end_cb`

Callback function associated to the group or NULL.

12.2.2.4 `adcerrorcallback_t ADCConversionGroup::error_cb`

Error callback or NULL.

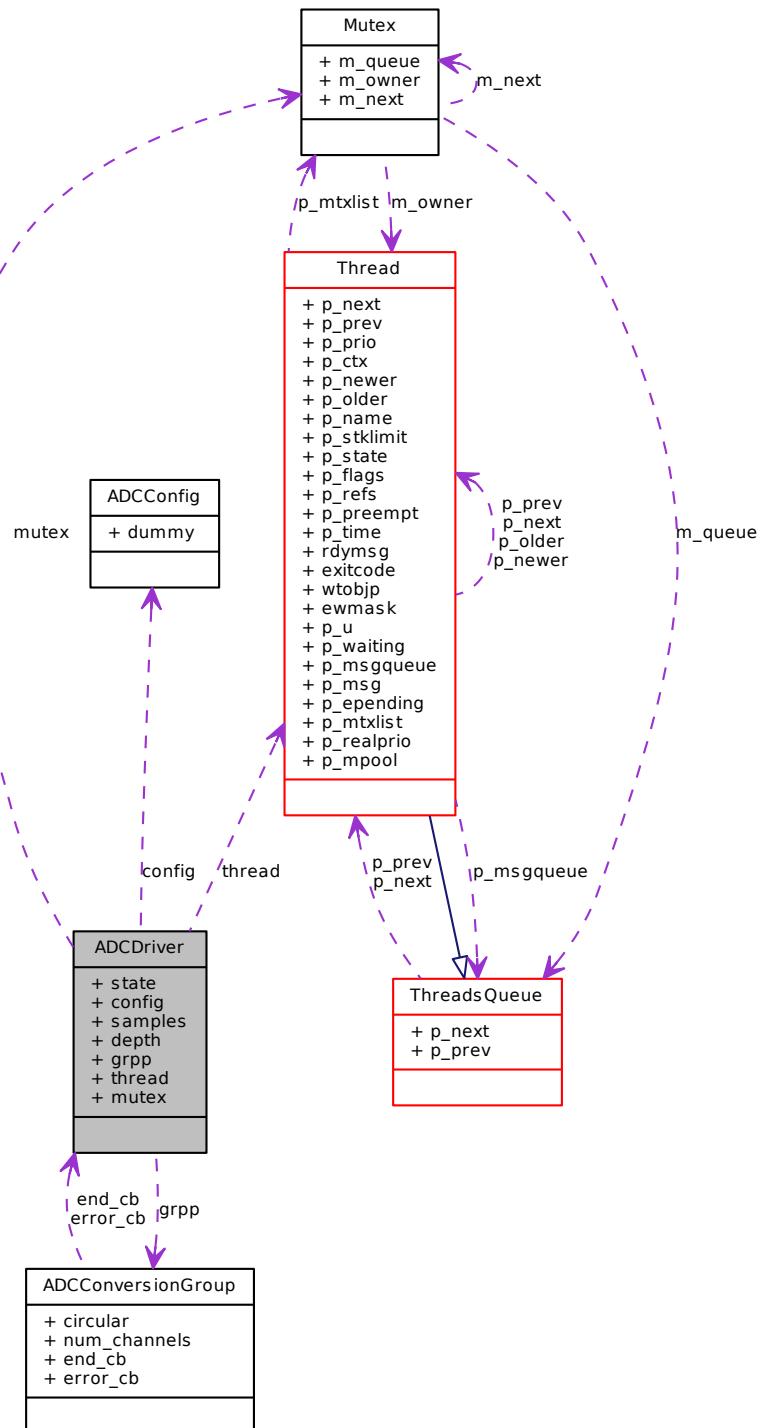
12.3 ADCDriver Struct Reference

12.3.1 Detailed Description

Structure representing an ADC driver.

```
#include <adc_llld.h>
```

Collaboration diagram for ADCDriver:



Data Fields

- [adcstate_t state](#)

Driver state.

- const ADCConfig * config
Current configuration data.
- adcsample_t * samples
Current samples buffer pointer or NULL.
- size_t depth
Current samples buffer depth or 0.
- const ADCConversionGroup * grpp
Current conversion group pointer or NULL.
- Thread * thread
Waiting thread.
- Mutex mutex
Mutex protecting the peripheral.

12.3.2 Field Documentation

12.3.2.1 adcstate_t ADCDriver::state

Driver state.

12.3.2.2 const ADCConfig* ADCDriver::config

Current configuration data.

12.3.2.3 adcsample_t* ADCDriver::samples

Current samples buffer pointer or NULL.

12.3.2.4 size_t ADCDriver::depth

Current samples buffer depth or 0.

12.3.2.5 const ADCConversionGroup* ADCDriver::grpp

Current conversion group pointer or NULL.

12.3.2.6 Thread* ADCDriver::thread

Waiting thread.

12.3.2.7 Mutex ADCDriver::mutex

Mutex protecting the peripheral.

12.4 BaseAsynchronousChannel Struct Reference

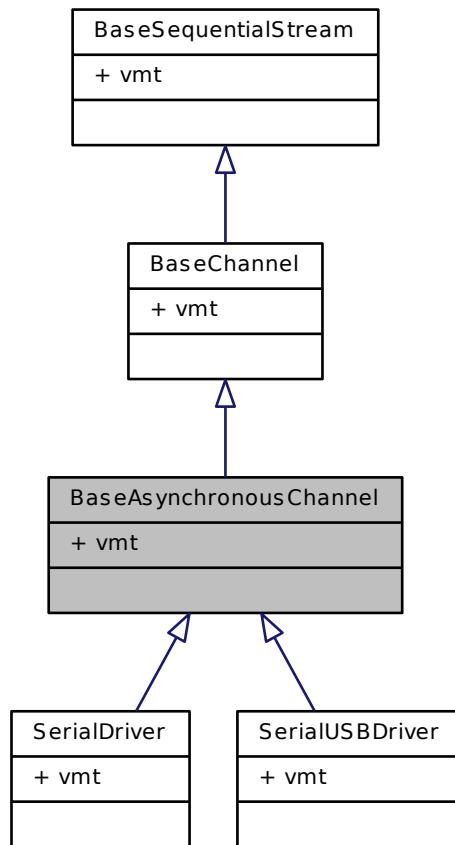
12.4.1 Detailed Description

Base asynchronous channel class.

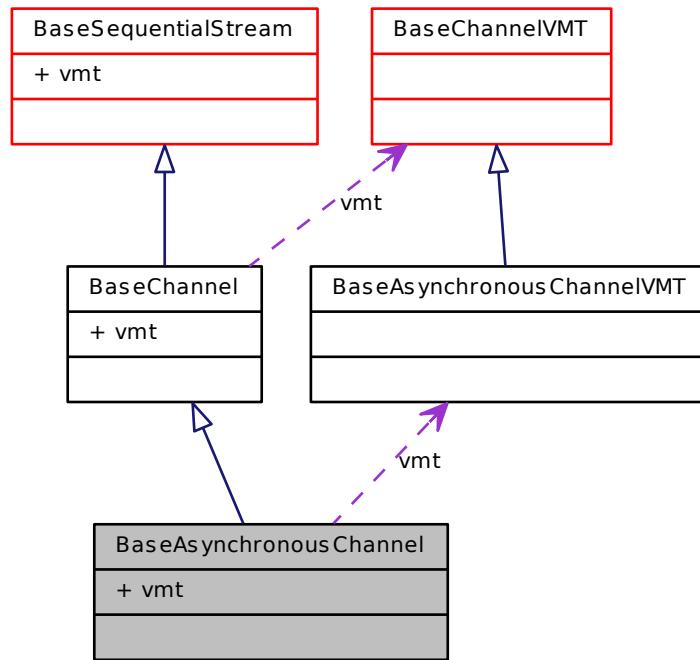
This class extends [BaseChannel](#) by adding event sources fields for asynchronous I/O for use in an event-driven environment.

```
#include <io_channel.h>
```

Inheritance diagram for BaseAsynchronousChannel:



Collaboration diagram for BaseAsynchronousChannel:



Data Fields

- struct [BaseAsynchronousChannelVMT](#) * `vmt`

Virtual Methods Table.

12.4.2 Field Documentation

12.4.2.1 struct [BaseAsynchronousChannelVMT](#)* `BaseAsynchronousChannel::vmt`

Virtual Methods Table.

Reimplemented from [BaseChannel](#).

Reimplemented in [SerialDriver](#), and [SerialUSBDriver](#).

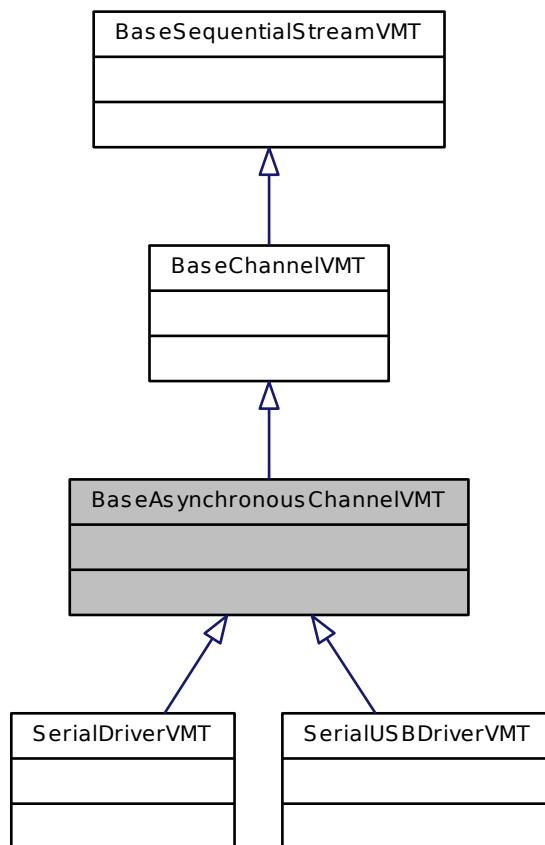
12.5 BaseAsynchronousChannelVMT Struct Reference

12.5.1 Detailed Description

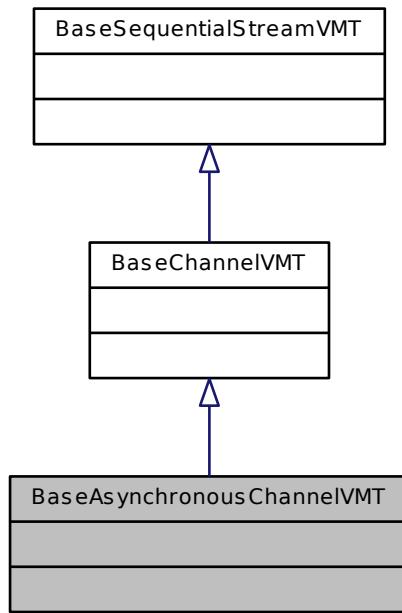
[BaseAsynchronousChannel](#) virtual methods table.

```
#include <io_channel.h>
```

Inheritance diagram for BaseAsynchronousChannelVMT:



Collaboration diagram for BaseAsynchronousChannelVMT:



12.6 BaseBlockDevice Struct Reference

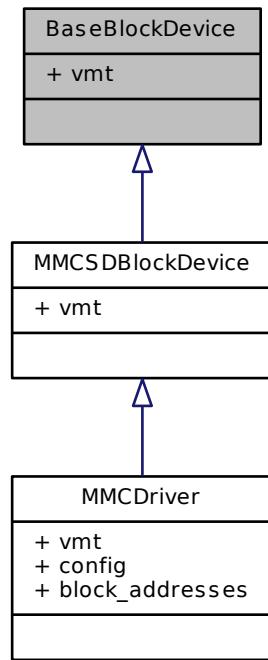
12.6.1 Detailed Description

Base block device class.

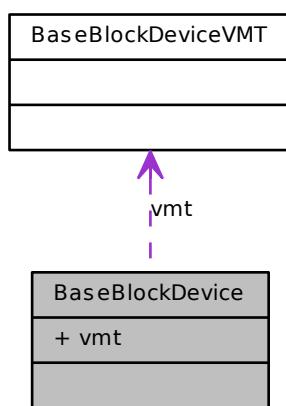
This class represents a generic, block-accessible, device.

```
#include <io_block.h>
```

Inheritance diagram for BaseBlockDevice:



Collaboration diagram for BaseBlockDevice:



Data Fields

- struct `BaseBlockDeviceVMT` * `vmt`

Virtual Methods Table.

12.6.2 Field Documentation

12.6.2.1 struct BaseBlockDeviceVMT* BaseBlockDevice::vmt

Virtual Methods Table.

Reimplemented in [MMCDriver](#), and [MMCSDBlockDevice](#).

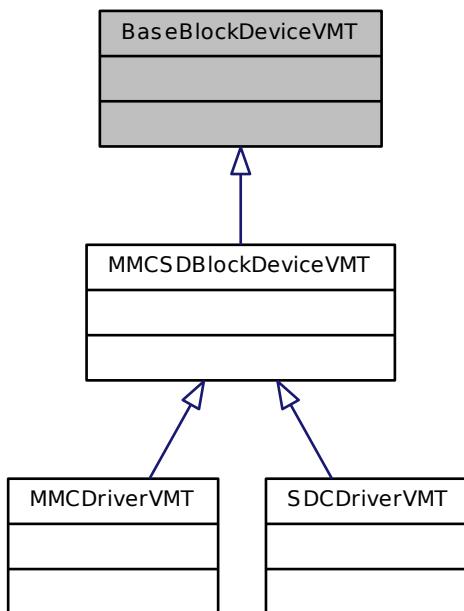
12.7 BaseBlockDeviceVMT Struct Reference

12.7.1 Detailed Description

[BaseBlockDevice](#) virtual methods table.

```
#include <io_block.h>
```

Inheritance diagram for BaseBlockDeviceVMT:



12.8 BaseChannel Struct Reference

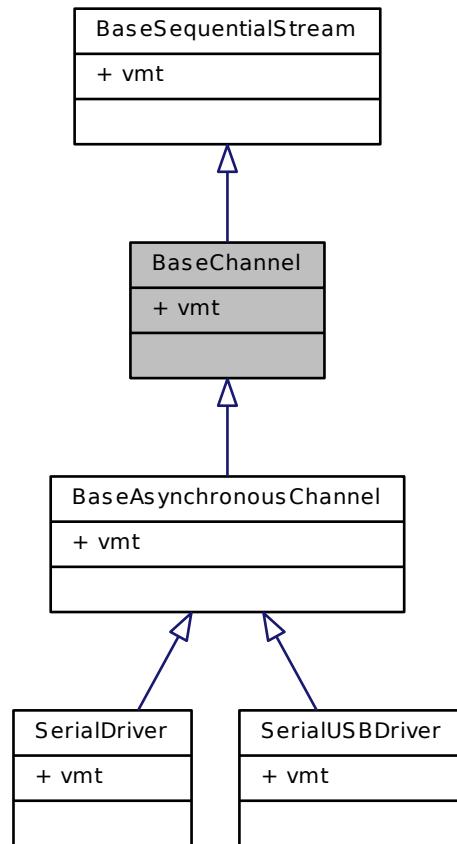
12.8.1 Detailed Description

Base channel class.

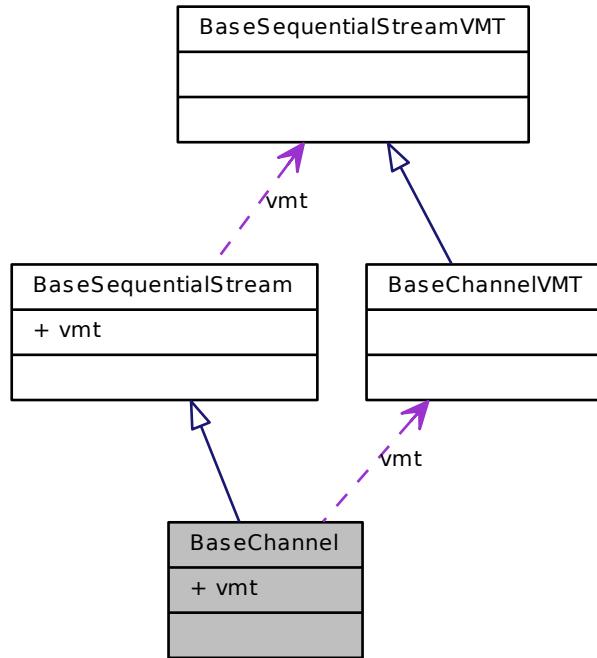
This class represents a generic, byte-wide, I/O channel. This class introduces generic I/O primitives with timeout specification.

```
#include <io_channel.h>
```

Inheritance diagram for BaseChannel:



Collaboration diagram for BaseChannel:



Data Fields

- struct [BaseChannelVMT](#) * `vmt`

Virtual Methods Table.

12.8.2 Field Documentation

12.8.2.1 struct [BaseChannelVMT](#)* `BaseChannel::vmt`

Virtual Methods Table.

Reimplemented from [BaseSequentialStream](#).

Reimplemented in [BaseAsynchronousChannel](#), [SerialDriver](#), and [SerialUSBDriver](#).

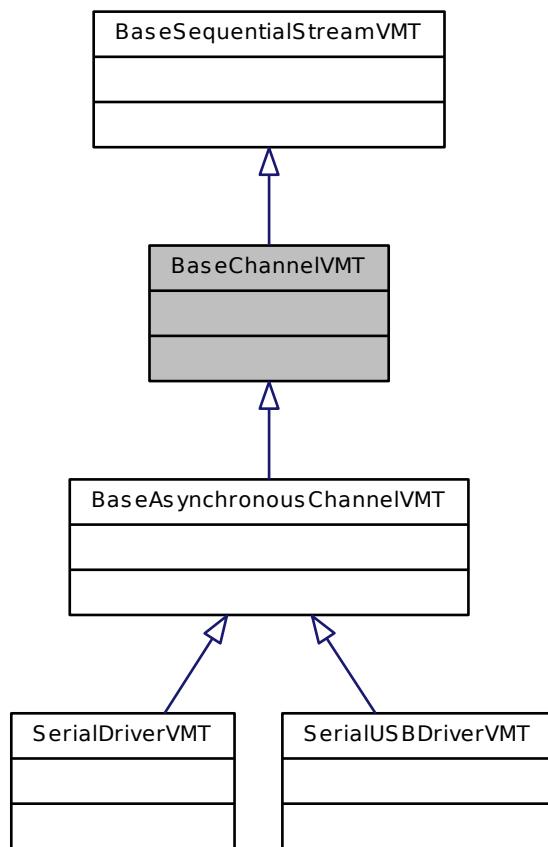
12.9 BaseChannelVMT Struct Reference

12.9.1 Detailed Description

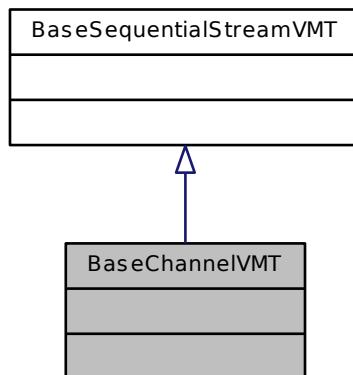
`BaseChannel` virtual methods table.

```
#include <io_channel.h>
```

Inheritance diagram for BaseChannelVMT:



Collaboration diagram for BaseChannelVMT:



12.10 BaseFileStream Struct Reference

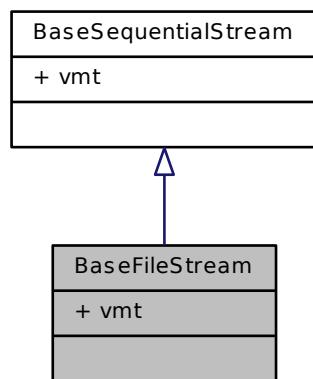
12.10.1 Detailed Description

Base file stream class.

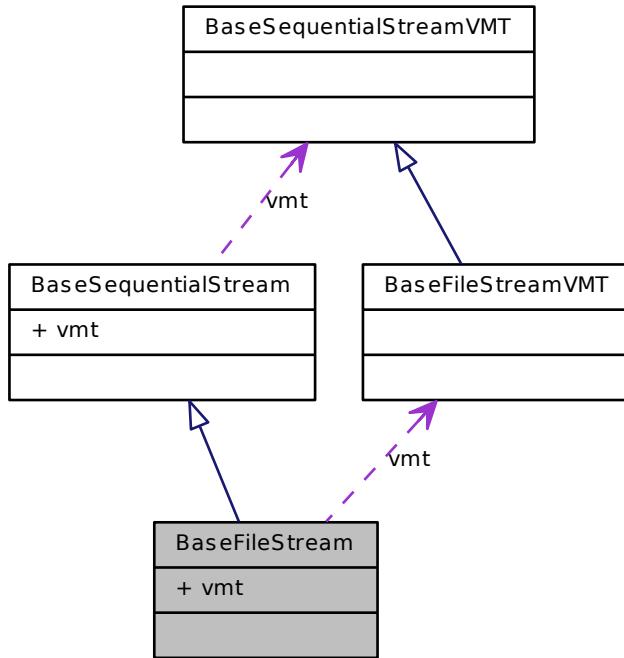
This class represents a generic file data stream.

```
#include <chfiles.h>
```

Inheritance diagram for BaseFileStream:



Collaboration diagram for BaseFileStream:



Data Fields

- struct [BaseFileStreamVMT](#) * **vmt**

Virtual Methods Table.

12.10.2 Field Documentation

12.10.2.1 struct [BaseFileStreamVMT](#)* **BaseFileStream::vmt**

Virtual Methods Table.

Reimplemented from [BaseSequentialStream](#).

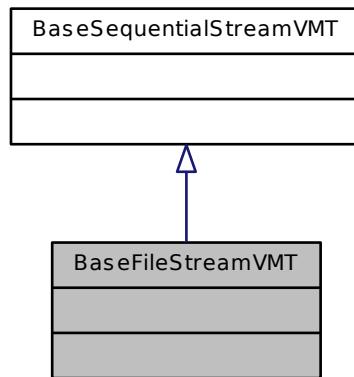
12.11 BaseFileStreamVMT Struct Reference

12.11.1 Detailed Description

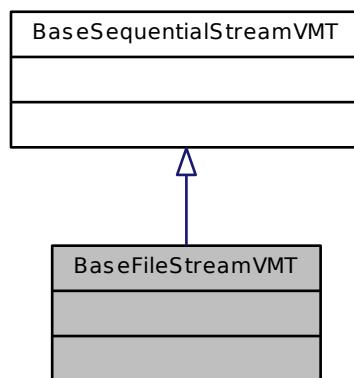
[BaseFileStream](#) virtual methods table.

```
#include <chfiles.h>
```

Inheritance diagram for BaseFileStreamVMT:



Collaboration diagram for BaseFileStreamVMT:



12.12 BaseSequentialStream Struct Reference

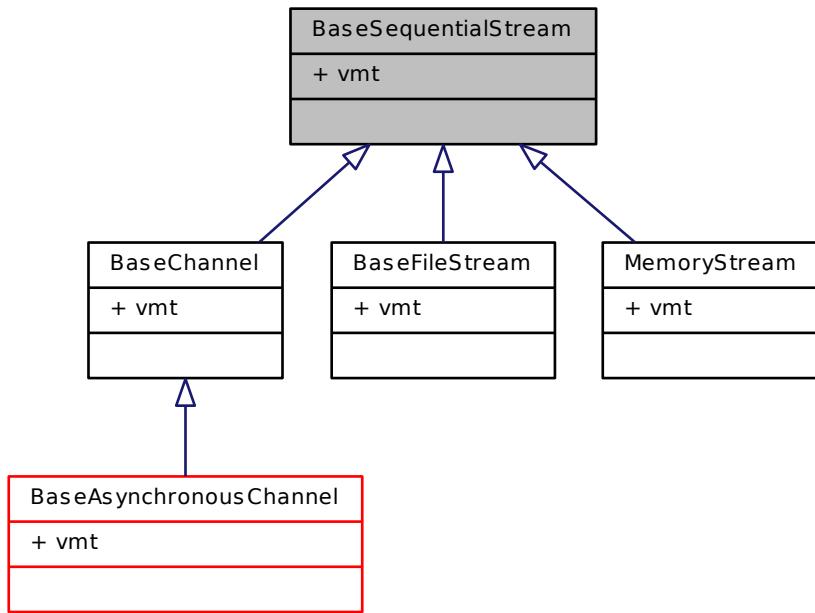
12.12.1 Detailed Description

Base stream class.

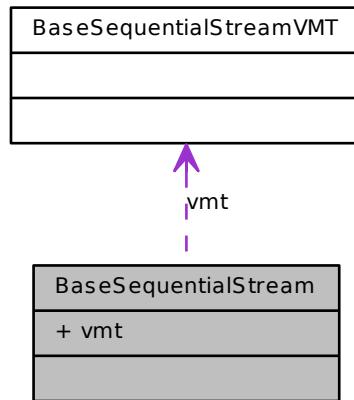
This class represents a generic blocking unbuffered sequential data stream.

```
#include <chstreams.h>
```

Inheritance diagram for BaseSequentialStream:



Collaboration diagram for BaseSequentialStream:



Data Fields

- struct `BaseSequentialStreamVMT` * `vmt`

Virtual Methods Table.

12.12.2 Field Documentation

12.12.2.1 struct BaseSequentialStreamVMT* BaseSequentialStream::vmt

Virtual Methods Table.

Reimplemented in [BaseFileStream](#), [BaseChannel](#), [BaseAsynchronousChannel](#), [SerialDriver](#), [SerialUSBDriver](#), and [MemoryStream](#).

12.13 chibios_rt::BaseSequentialStreamInterface Class Reference

12.13.1 Detailed Description

Interface of a [BaseSequentialStream](#).

Note

You can cast a [BaseSequentialStream](#) to this interface and use it, the memory layout is the same.

```
#include <ch.hpp>
```

Public Member Functions

- virtual size_t [write](#) (const uint8_t *bp, size_t n)=0
Sequential Stream write.
- virtual size_t [read](#) (uint8_t *bp, size_t n)=0
Sequential Stream read.
- virtual msg_t [put](#) (uint8_t b)=0
Sequential Stream blocking byte write.
- virtual msg_t [get](#) (void)=0
Sequential Stream blocking byte read.

12.13.2 Member Function Documentation

12.13.2.1 virtual size_t chibios_rt::BaseSequentialStreamInterface::write (const uint8_t * bp, size_t n) [pure virtual]

Sequential Stream write.

The function writes data from a buffer to a stream.

Parameters

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.

12.13.2.2 virtual size_t chibios_rt::BaseSequentialStreamInterface::read (uint8_t * *bp*, size_t *n*) [pure virtual]

Sequential Stream read.

The function reads data from a stream into a buffer.

Parameters

<i>out</i>	<i>bp</i>	pointer to the data buffer
<i>in</i>	<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.

12.13.2.3 virtual msg_t chibios_rt::BaseSequentialStreamInterface::put (uint8_t *b*) [pure virtual]

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

<i>in</i>	<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.

12.13.2.4 virtual msg_t chibios_rt::BaseSequentialStreamInterface::get (void) [pure virtual]

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.

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.

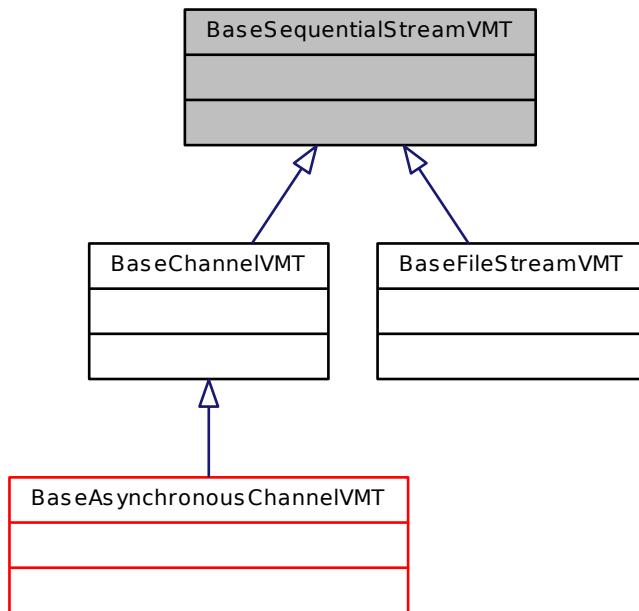
12.14 BaseSequentialStreamVMT Struct Reference

12.14.1 Detailed Description

`BaseSequentialStream` virtual methods table.

```
#include <chstreams.h>
```

Inheritance diagram for `BaseSequentialStreamVMT`:



12.15 chibios_rt::BaseStaticThread< N > Class Template Reference

12.15.1 Detailed Description

```
template<int N>class chibios_rt::BaseStaticThread< N >
```

Static threads template class.

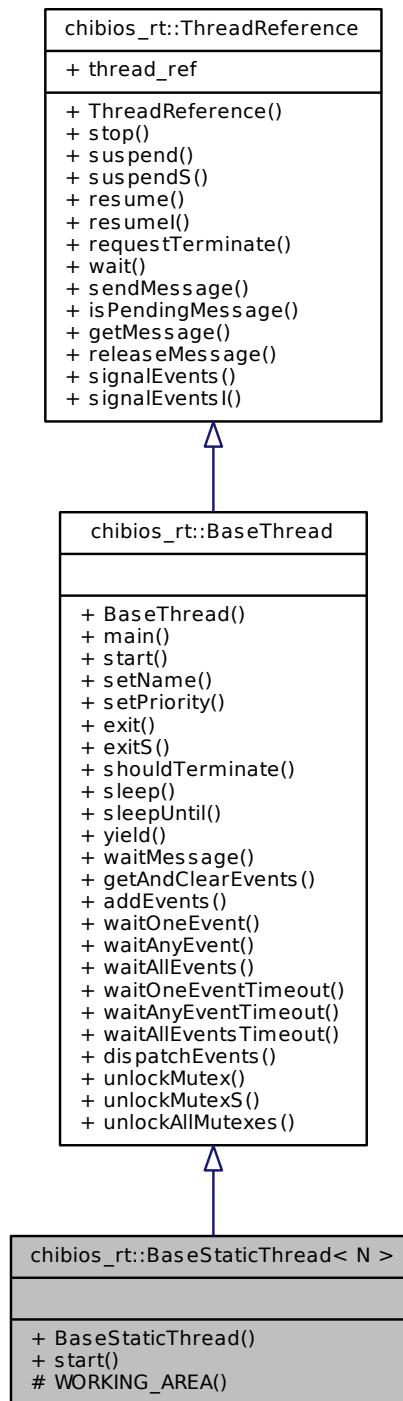
This class introduces static working area allocation.

Parameters

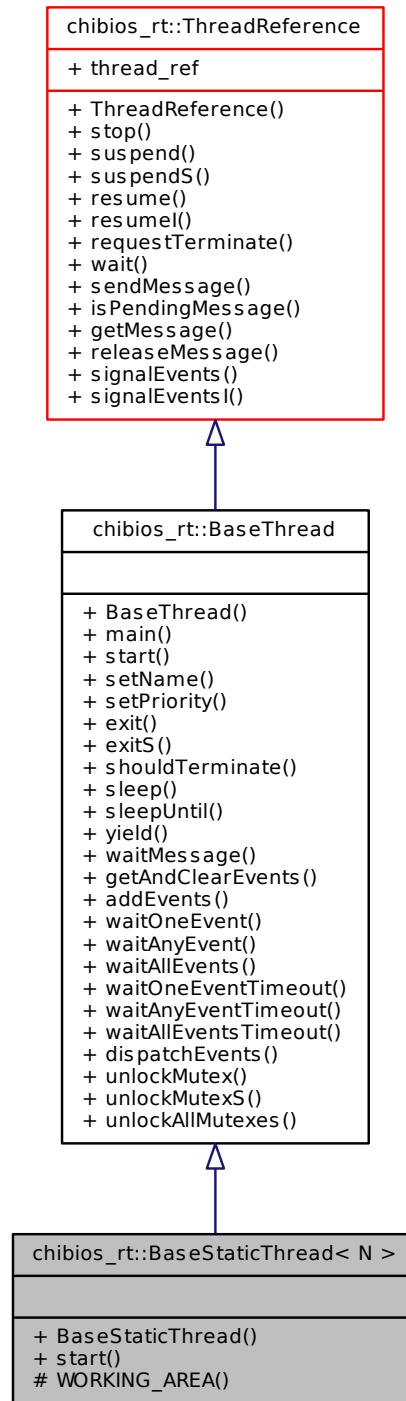
N the working area size for the thread class

```
#include <ch.hpp>
```

Inheritance diagram for chibios_rt::BaseStaticThread< N >:



Collaboration diagram for chibios_rt::BaseStaticThread< N >:



Public Member Functions

- **BaseStaticThread (void)**

Thread constructor.

- virtual [ThreadReference start \(tprio_t prio\)](#)

Creates and starts a system thread.

12.15.2 Constructor & Destructor Documentation

12.15.2.1 template<int N> [chibios_rt::BaseStaticThread< N >::BaseStaticThread \(void \) \[inline\]](#)

[Thread](#) constructor.

The thread object is initialized but the thread is not started here.

Function Class:

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

12.15.3 Member Function Documentation

12.15.3.1 template<int N> virtual [ThreadReference chibios_rt::BaseStaticThread< N >::start \(tprio_t prio \) \[inline, virtual\]](#)

Creates and starts a system thread.

Parameters

in *prio* thread priority

Returns

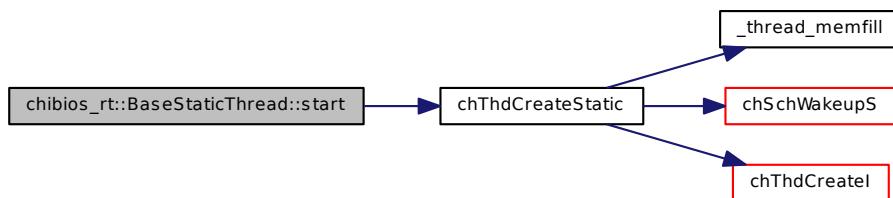
A reference to the created thread with reference counter set to one.

Function Class:

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

Reimplemented from [chibios_rt::BaseThread](#).

Here is the call graph for this function:



12.16 chibios_rt::BaseThread Class Reference

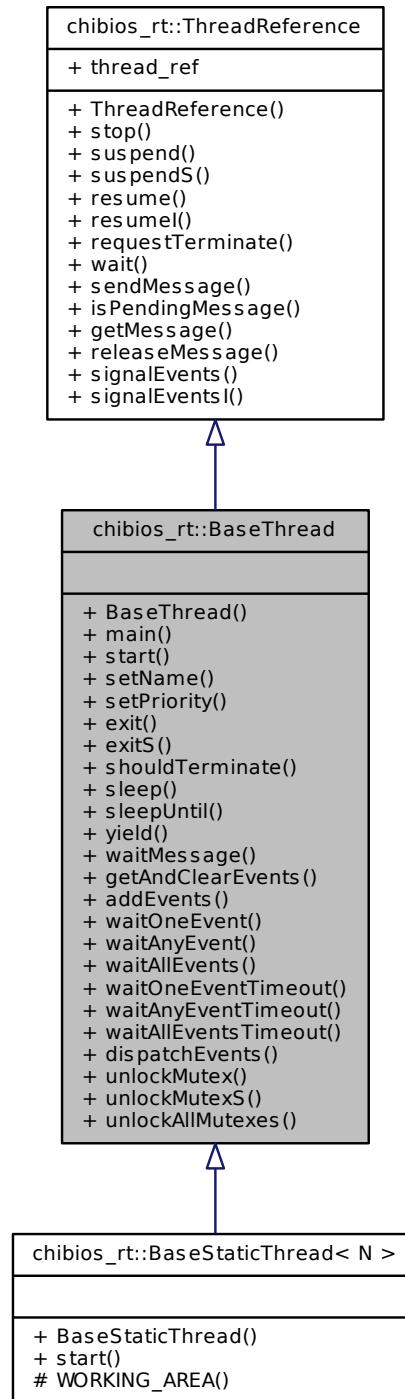
12.16.1 Detailed Description

Abstract base class for a ChibiOS/RT thread.

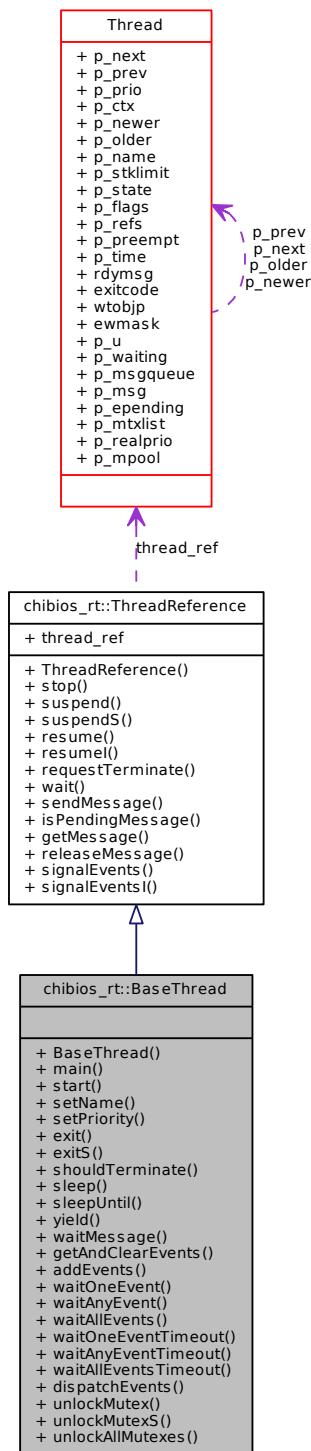
The thread body is the virtual function `Main()`.

```
#include <ch.hpp>
```

Inheritance diagram for chibios_rt::BaseThread:



Collaboration diagram for chibios_rt::BaseThread:



Public Member Functions

- **BaseThread (void)**
BaseThread constructor.

- virtual `msg_t main (void)`
Thread body function.
- virtual `ThreadReference start (tprio_t prio)`
Creates and starts a system thread.

Static Public Member Functions

- static void `setName (const char *tname)`
Sets the current thread name.
- static `tprio_t setPriority (tprio_t newprio)`
Changes the running thread priority level then reschedules if necessary.
- static void `exit (msg_t msg)`
Terminates the current thread.
- static void `exitS (msg_t msg)`
Terminates the current thread.
- static bool `shouldTerminate (void)`
Verifies if the current thread has a termination request pending.
- static void `sleep (systime_t interval)`
Suspends the invoking thread for the specified time.
- static void `sleepUntil (systime_t time)`
Suspends the invoking thread until the system time arrives to the specified value.
- static void `yield (void)`
Yields the time slot.
- static `ThreadReference waitMessage (void)`
Waits for a message.
- static `eventmask_t getAndClearEvents (eventmask_t mask)`
Clears the pending events specified in the mask.
- static `eventmask_t addEvents (eventmask_t mask)`
*Adds (OR) a set of event flags on the current thread, this is **much** faster than using `chEvtBroadcast ()` or `chEvtSignal ()`.*
- static `eventmask_t waitOneEvent (eventmask_t ewmask)`
Waits for a single event.
- static `eventmask_t waitAnyEvent (eventmask_t ewmask)`
Waits for any of the specified events.
- static `eventmask_t waitAllEvents (eventmask_t ewmask)`
Waits for all the specified event flags then clears them.
- static `eventmask_t waitOneEventTimeout (eventmask_t ewmask, systime_t time)`
Waits for a single event.
- static `eventmask_t waitAnyEventTimeout (eventmask_t ewmask, systime_t time)`
Waits for any of the specified events.
- static `eventmask_t waitAllEventsTimeout (eventmask_t ewmask, systime_t time)`
Waits for all the specified event flags then clears them.
- static void `dispatchEvents (const evhandler_t handlers[], eventmask_t mask)`
Invokes the event handlers associated to an event flags mask.
- static void `unlockMutex (void)`
Unlocks the next owned mutex in reverse lock order.
- static void `unlockMutexS (void)`
Unlocks the next owned mutex in reverse lock order.
- static void `unlockAllMutexes (void)`
Unlocks all the mutexes owned by the invoking thread.

12.16.2 Constructor & Destructor Documentation

12.16.2.1 `chibios_rt::BaseThread::BaseThread(void)`

`BaseThread` constructor.

Function Class:

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

12.16.3 Member Function Documentation

12.16.3.1 `msg_t chibios_rt::BaseThread::main(void) [virtual]`

`Thread` body function.

Returns

The exit message.

Function Class:

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

12.16.3.2 `ThreadReference chibios_rt::BaseThread::start(tprio_t prio) [virtual]`

Creates and starts a system thread.

Parameters

in *prio* thread priority

Returns

A reference to the created thread with reference counter set to one.

Function Class:

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

Reimplemented in `chibios_rt::BaseStaticThread< N >`.

12.16.3.3 `void chibios_rt::BaseThread::setName(const char * tname) [static]`

Sets the current thread name.

Precondition

This function only stores the pointer to the name if the option CH_USE_REGISTRY is enabled else no action is performed.

Parameters

in *tname* 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.

12.16.3.4 `tprio_t chibios_rt::BaseThread::setPriority(tprio_t newprio) [static]`

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 *newprio* 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:

12.16.3.5 `void chibios_rt::BaseThread::exit(msg_t msg) [static]`

Terminates the current thread.

The thread goes in the THD_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:

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:



12.16.3.6 void chibios_rt::BaseThread::exitS (msg_t msg) [static]

Terminates the current thread.

The thread goes in the THD_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.

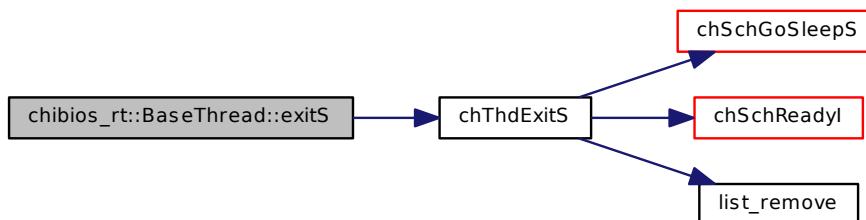
Parameters

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:



12.16.3.7 bool chibios_rt::BaseThread::shouldTerminate (void) [static]

Verifies if the current thread has a termination request pending.

Note

Can be invoked in any context.

Return values

TRUE termination request pending.
FALSE termination request not pending.

Function Class:

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

12.16.3.8 void chibios_rt::BaseThread::sleep(systime_t interval) [static]

Suspends the invoking thread for the specified time.

Parameters

- in *interval* the delay in system ticks, the special values are handled as follow:
 - *TIME_INFINITE* the thread enters an infinite sleep state.
 - *TIME_IMMEDIATE* 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:

**12.16.3.9 void chibios_rt::BaseThread::sleepUntil(systime_t time) [static]**

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

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:



12.16.3.10 void chibios_rt::BaseThread::yield(void) [static]

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:



12.16.3.11 ThreadReference chibios_rt::BaseThread::waitMessage(void) [static]

Waits for a message.

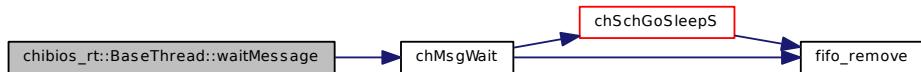
Returns

The sender 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:



12.16.3.12 eventmask_t chibios_rt::BaseThread::getAndClearEvents (eventmask_t mask) [static]

Clears the pending events specified in the mask.

Parameters

in *mask* 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:

**12.16.3.13 eventmask_t chibios_rt::BaseThread::addEvents (eventmask_t mask) [static]**

Adds (OR) a set of event flags on the current thread, this is **much** faster than using [chEvtBroadcast\(\)](#) or [chEvtSignal\(\)](#).

Parameters

in *mask* the event flags to be added

Returns

The current pending events 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:



12.16.3.14 eventmask_t chibios_rt::BaseThread::waitOneEvent(eventmask_t ewmask) [static]

Waits for a single event.

A pending event among those specified in `ewmask` is selected, cleared and its mask 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>ewmask</code> mask of the events that the function should wait for, <code>ALL_EVENTS</code> enables all the events
----	--

Returns

The mask of the lowest id served and cleared event.

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:



12.16.3.15 eventmask_t chibios_rt::BaseThread::waitAnyEvent(eventmask_t ewmask) [static]

Waits for any of the specified events.

The function waits for any event among those specified in `ewmask` to become pending then the events are cleared and returned.

Parameters

in	<code>ewmask</code> mask of the 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:



12.16.3.16 eventmask_t chibios_rt::BaseThread::waitAllEvents (eventmask_t ewmask) [static]

Waits for all the specified event flags then clears them.

The function waits for all the events specified in `ewmask` to become pending then the events are cleared and returned.

Parameters

in `ewmask` mask of the event ids that the function should wait for

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:



12.16.3.17 eventmask_t chibios_rt::BaseThread::waitOneEventTimeout (eventmask_t ewmask, systime_t time) [static]

Waits for a single event.

A pending event among those specified in `ewmask` is selected, cleared and its mask 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 a higher priority.

Parameters

in `ewmask` mask of the events that the function should wait for, `ALL_EVENTS` enables all the events
 in `time` the number of ticks before the operation timeouts

Returns

The mask of the lowest id served and cleared event.

Return values

0 if the specified timeout expired.

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:



12.16.3.18 `eventmask_t chibios_rt::BaseThread::waitAnyEventTimeout(eventmask_t ewmask, systime_t time) [static]`

Waits for any of the specified events.

The function waits for any event among those specified in `ewmask` to become pending then the events are cleared and returned.

Parameters

in	<code>ewmask</code>	mask of the events that the function should wait for, <code>ALL_EVENTS</code> enables all the events
in	<code>time</code>	the number of ticks before the operation timeouts

Returns

The mask of the served and cleared events.

Return values

0 if the specified timeout expired.

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:



12.16.3.19 eventmask_t chibios_rt::BaseThread::waitAllEventsTimeout (eventmask_t *ewmask*, systime_t *time*) [static]

Waits for all the specified event flags then clears them.

The function waits for all the events specified in *ewmask* to become pending then the events are cleared and returned.

Parameters

in	<i>ewmask</i>	mask of the event ids that the function should wait for
in	<i>time</i>	the number of ticks before the operation timeouts

Returns

The mask of the served and cleared events.

Return values

0 if the specified timeout expired.

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:



12.16.3.20 void chibios_rt::BaseThread::dispatchEvents (const evhandler_t *handlers*[], eventmask_t *mask*) [static]

Invokes the event handlers associated to an event flags mask.

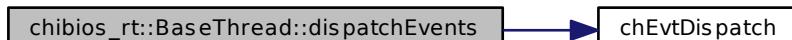
Parameters

in	<i>mask</i>	mask of the event flags to be dispatched
in	<i>handlers</i>	an array of evhandler_t. The array must have size equal to the number of bits in eventmask_t.

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:



12.16.3.21 void chibios_rt::BaseThread::unlockMutex (void) [static]

Unlocks the next owned mutex in reverse lock order.

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.

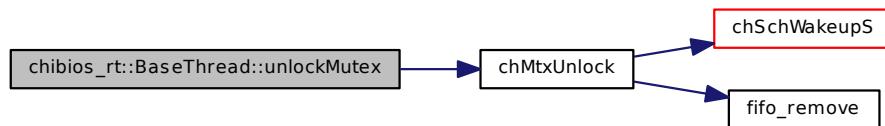
Returns

A pointer to the unlocked mutex.

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:



12.16.3.22 void chibios_rt::BaseThread::unlockMutexS (void) [static]

Unlocks the next owned mutex in reverse lock order.

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.

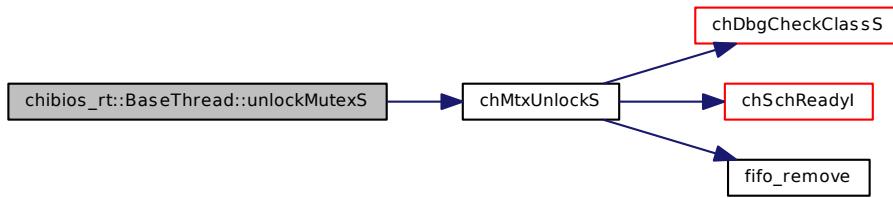
Returns

A pointer to the unlocked mutex.

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:



12.16.3.23 void chibios_rt::BaseThread::unlockAllMutexes (void) [static]

Unlocks all the mutexes owned by the invoking thread.

Postcondition

The stack of owned mutexes is emptied and all the found mutexes are unlocked.

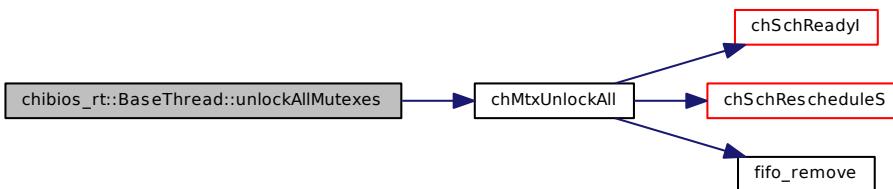
Note

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:



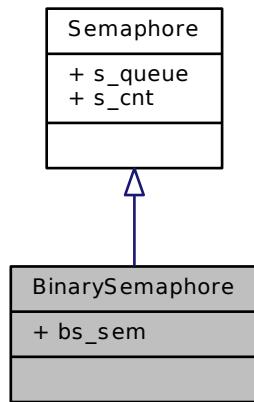
12.17 BinarySemaphore Struct Reference

12.17.1 Detailed Description

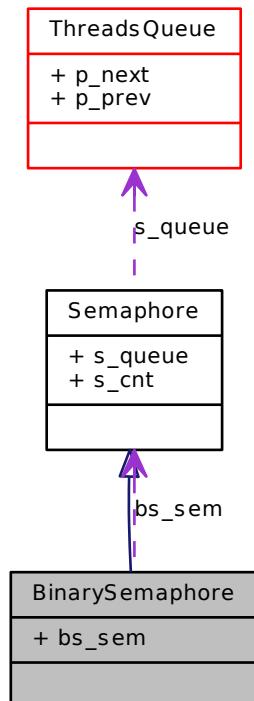
Binary semaphore type.

```
#include <chbsem.h>
```

Inheritance diagram for BinarySemaphore:



Collaboration diagram for BinarySemaphore:



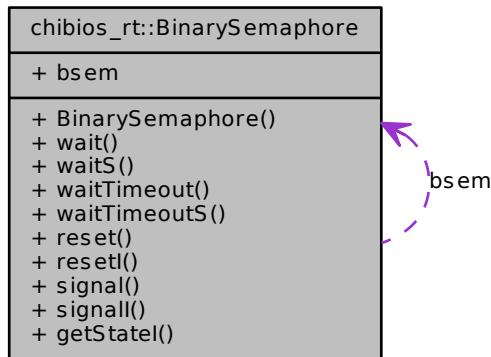
12.18 chibios_rt::BinarySemaphore Class Reference

12.18.1 Detailed Description

Class encapsulating a binary semaphore.

```
#include <ch.hpp>
```

Collaboration diagram for chibios_rt::BinarySemaphore:



Public Member Functions

- **BinarySemaphore (bool taken)**
BinarySemaphore constructor.
- **msg_t wait (void)**
Wait operation on the binary semaphore.
- **msg_t waitS (void)**
Wait operation on the binary semaphore.
- **msg_t waitTimeout (systime_t time)**
Wait operation on the binary semaphore.
- **msg_t waitTimeoutS (systime_t time)**
Wait operation on the binary semaphore.
- **void reset (bool taken)**
Reset operation on the binary semaphore.
- **void resetI (bool taken)**
Reset operation on the binary semaphore.
- **void signal (void)**
Performs a signal operation on a binary semaphore.
- **void signall (void)**
Performs a signal operation on a binary semaphore.
- **bool getStateI (void)**
Returns the binary semaphore current state.

Data Fields

- `::BinarySemaphore bsem`
`Embedded Semaphore structure.`

12.18.2 Constructor & Destructor Documentation

12.18.2.1 BinarySemaphore::BinarySemaphore (bool taken)

`BinarySemaphore` constructor.

The embedded `BinarySemaphore` structure is initialized.

Parameters

- | | |
|-----------------|---|
| <code>in</code> | <code>taken</code> initial state of the binary semaphore: |
| | <ul style="list-style-type: none"> • <code>false</code>, the initial state is not taken. • <code>true</code>, the initial state is taken. |

Function Class:

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

12.18.3 Member Function Documentation

12.18.3.1 msg_t BinarySemaphore::wait (void)

Wait operation on the binary semaphore.

Returns

A message specifying how the invoking thread has been released from the semaphore.

Return values

- | | |
|------------------------|--|
| <code>RDY_OK</code> | if the binary semaphore has been successfully taken. |
| <code>RDY_RESET</code> | 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.

12.18.3.2 msg_t BinarySemaphore::waitS (void)

Wait operation on the binary semaphore.

Returns

A message specifying how the invoking thread has been released from the semaphore.

Return values

- | | |
|------------------------|--|
| <code>RDY_OK</code> | if the binary semaphore has been successfully taken. |
| <code>RDY_RESET</code> | 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.

12.18.3.3 `msg_t BinarySemaphore::waitTimeout(systime_t time)`

Wait operation on the binary semaphore.

Parameters

- in *time* the number of ticks before the operation timeouts, the following special values are allowed:
- *TIME_IMMEDIATE* immediate timeout.
 - *TIME_INFINITE* no timeout.

Returns

A message specifying how the invoking thread has been released from the semaphore.

Return values

- | | |
|--------------------|--|
| <i>RDY_OK</i> | if the binary semaphore has been successfully taken. |
| <i>RDY_RESET</i> | if the binary semaphore has been reset using <code>bsemReset()</code> . |
| <i>RDY_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.

12.18.3.4 `msg_t BinarySemaphore::waitForS(systime_t time)`

Wait operation on the binary semaphore.

Parameters

- in *time* the number of ticks before the operation timeouts, the following special values are allowed:
- *TIME_IMMEDIATE* immediate timeout.
 - *TIME_INFINITE* no timeout.

Returns

A message specifying how the invoking thread has been released from the semaphore.

Return values

- | | |
|--------------------|--|
| <i>RDY_OK</i> | if the binary semaphore has been successfully taken. |
| <i>RDY_RESET</i> | if the binary semaphore has been reset using <code>bsemReset()</code> . |
| <i>RDY_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.

12.18.3.5 `void BinarySemaphore::reset(bool taken)`

Reset operation on the binary semaphore.

Note

The released threads can recognize they were wakened up by a reset rather than a signal because the `bsemWait()` will return *RDY_RESET* instead of *RDY_OK*.

Parameters

- in *taken* new state of the binary semaphore
- *FALSE*, the new state is not taken.
 - *TRUE*, 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.

12.18.3.6 void BinarySemaphore::reset(bool *taken*)

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 `bsemWait()` will return `RDY_RESET` instead of `RDY_OK`.
This function does not reschedule.

Parameters

- in *taken* new state of the binary semaphore
- *FALSE*, the new state is not taken.
 - *TRUE*, 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.

12.18.3.7 void BinarySemaphore::signal(void)

Performs a signal operation on a binary semaphore.

Function Class:

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

12.18.3.8 void BinarySemaphore::signall(void)

Performs a signal operation on a binary semaphore.

Note

This function does not reschedule.

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.

12.18.3.9 bool BinarySemaphore::getState(void)

Returns the binary semaphore current state.

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.

12.18.4 Field Documentation

12.18.4.1 ::BinarySemaphore chibios_rt::BinarySemaphore::bsem

Embedded [Semaphore](#) structure.

12.19 BlockDeviceInfo Struct Reference

12.19.1 Detailed Description

Block device info.

```
#include <io_block.h>
```

Data Fields

- `uint32_t blk_size`
Block size in bytes.
- `uint32_t blk_num`
Total number of blocks.

12.19.2 Field Documentation

12.19.2.1 `uint32_t BlockDeviceInfo::blk_size`

Block size in bytes.

12.19.2.2 `uint32_t BlockDeviceInfo::blk_num`

Total number of blocks.

12.20 CANConfig Struct Reference

12.20.1 Detailed Description

Driver configuration structure.

Note

Implementations may extend this structure to contain more, architecture dependent, fields.
It could be empty on some architectures.

```
#include <can_lld.h>
```

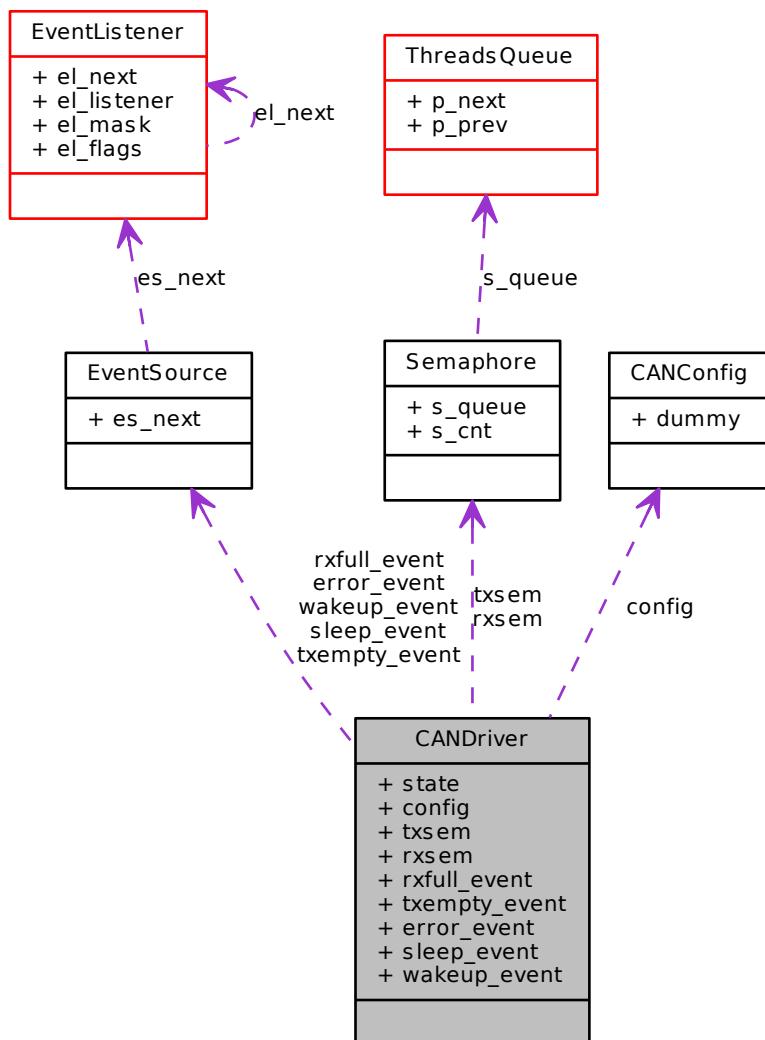
12.21 CANDriver Struct Reference

12.21.1 Detailed Description

Structure representing an CAN driver.

```
#include <can_lld.h>
```

Collaboration diagram for CANDriver:



Data Fields

- `canstate_t state`
Driver state.
- `const CANConfig * config`
Current configuration data.
- `Semaphore txsem`
Transmission queue semaphore.
- `Semaphore rxsem`
Receive queue semaphore.
- `EventSource rxfull_event`
One or more frames become available.
- `EventSource txempty_event`
One or more transmission mailbox become available.
- `EventSource error_event`
A CAN bus error happened.
- `EventSource sleep_event`
Entering sleep state event.
- `EventSource wakeup_event`
Exiting sleep state event.

12.21.2 Field Documentation

12.21.2.1 `canstate_t CANDriver::state`

Driver state.

12.21.2.2 `const CANConfig* CANDriver::config`

Current configuration data.

12.21.2.3 `Semaphore CANDriver::txsem`

Transmission queue semaphore.

12.21.2.4 `Semaphore CANDriver::rxsem`

Receive queue semaphore.

12.21.2.5 `EventSource CANDriver::rxfull_event`

One or more frames become available.

Note

After broadcasting this event it will not be broadcasted again until the received frames queue has been completely emptied. It is **not** broadcasted for each received frame. It is responsibility of the application to empty the queue by repeatedly invoking `chReceive()` when listening to this event. This behavior minimizes the interrupt served by the system because CAN traffic.

The flags associated to the listeners will indicate which receive mailboxes become non-empty.

12.21.2.6 EventSource CANDriver::txempty_event

One or more transmission mailbox become available.

Note

The flags associated to the listeners will indicate which transmit mailboxes become empty.

12.21.2.7 EventSource CANDriver::error_event

A CAN bus error happened.

Note

The flags associated to the listeners will indicate the error(s) that have occurred.

12.21.2.8 EventSource CANDriver::sleep_event

Entering sleep state event.

12.21.2.9 EventSource CANDriver::wakeup_event

Exiting sleep state event.

12.22 CANFilter Struct Reference

12.22.1 Detailed Description

CAN filter.

Note

Implementations may extend this structure to contain more, architecture dependent, fields.
It could not be present on some architectures.

```
#include <can_ll.h>
```

12.23 CANRxFrame Struct Reference

12.23.1 Detailed Description

CAN received frame.

Note

Accessing the frame data as word16 or word32 is not portable because machine data endianness, it can be still useful for a quick filling.

```
#include <can_ll.h>
```

12.23.2 Field Documentation

12.23.2.1 uint8_t CANRxFrame::DLC

Data length.

12.23.2.2 uint8_t CANRxFrame::RTR

Frame type.

12.23.2.3 uint8_t CANRxFrame::IDE

Identifier type.

12.23.2.4 uint32_t CANRxFrame::SID

Standard identifier.

12.23.2.5 uint32_t CANRxFrame::EID

Extended identifier.

12.23.2.6 uint8_t CANRxFrame::data8[8]

Frame data.

12.23.2.7 uint16_t CANRxFrame::data16[4]

Frame data.

12.23.2.8 uint32_t CANRxFrame::data32[2]

Frame data.

12.24 CANTxFrame Struct Reference

12.24.1 Detailed Description

CAN transmission frame.

Note

Accessing the frame data as word16 or word32 is not portable because machine data endianness, it can be still useful for a quick filling.

```
#include <can_lld.h>
```

12.24.2 Field Documentation

12.24.2.1 uint8_t CANTxFrame::DLC

Data length.

12.24.2.2 uint8_t CANTxFrame::RTR

Frame type.

12.24.2.3 uint8_t CANTxFrame::IDE

Identifier type.

12.24.2.4 uint32_t CANTxFrame::SID

Standard identifier.

12.24.2.5 uint32_t CANTxFrame::EID

Extended identifier.

12.24.2.6 uint8_t CANTxFrame::data8[8]

Frame data.

12.24.2.7 uint16_t CANTxFrame::data16[4]

Frame data.

12.24.2.8 uint32_t CANTxFrame::data32[2]

Frame data.

12.25 cdc_linecoding_t Struct Reference

12.25.1 Detailed Description

Type of Line Coding structure.

```
#include <serial_usb.h>
```

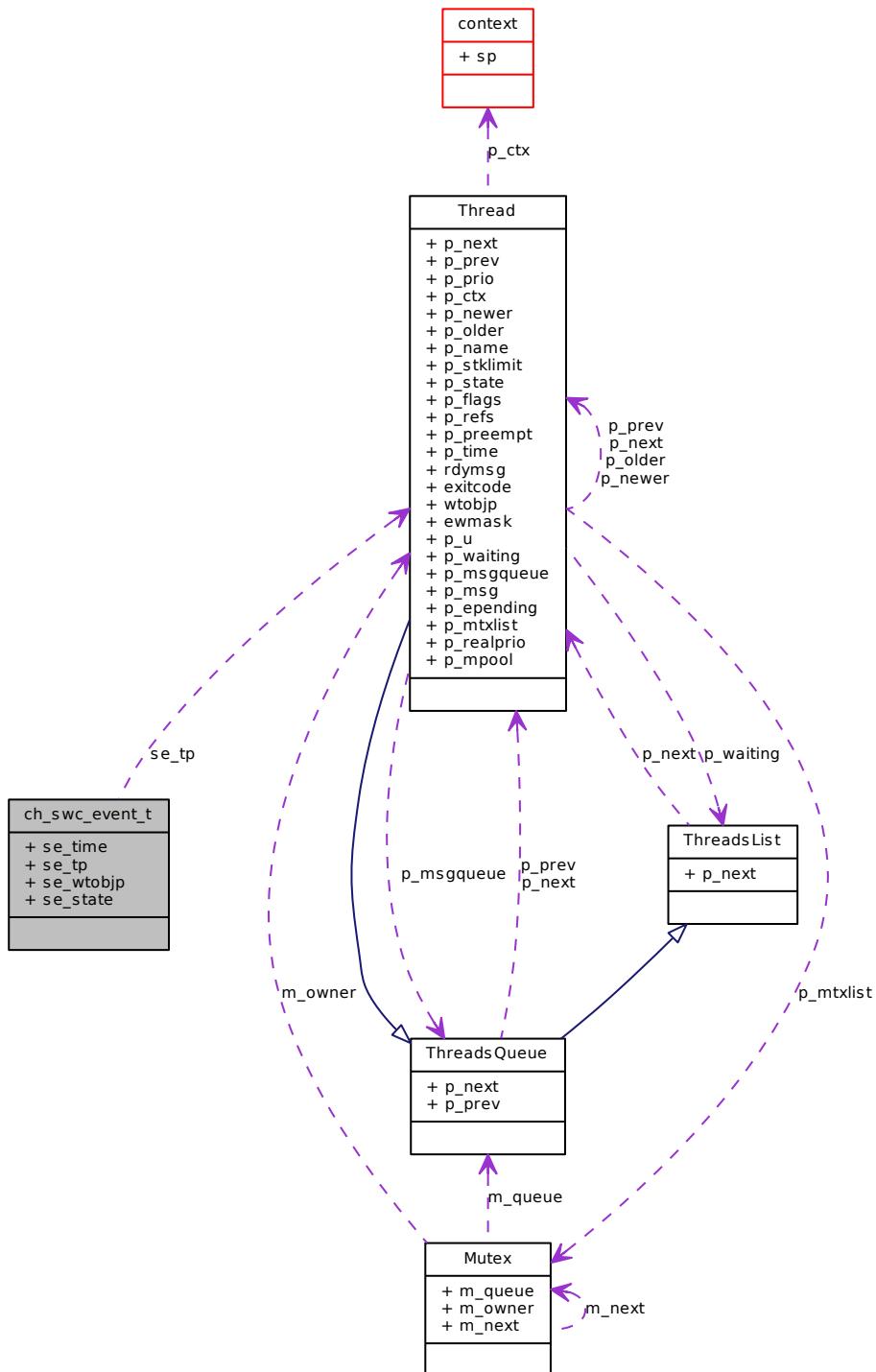
12.26 ch_swc_event_t Struct Reference

12.26.1 Detailed Description

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 * se_tp`
Switched in thread.
- `void * se_wtobjp`
Object where going to sleep.
- `uint8_t se_state`
Switched out thread state.

12.26.2 Field Documentation

12.26.2.1 systime_t ch_swc_event_t::se_time

Time of the switch event.

12.26.2.2 Thread* ch_swc_event_t::se_tp

Switched in thread.

12.26.2.3 void* ch_swc_event_t::se_wtobjp

Object where going to sleep.

12.26.2.4 uint8_t ch_swc_event_t::se_state

Switched out thread state.

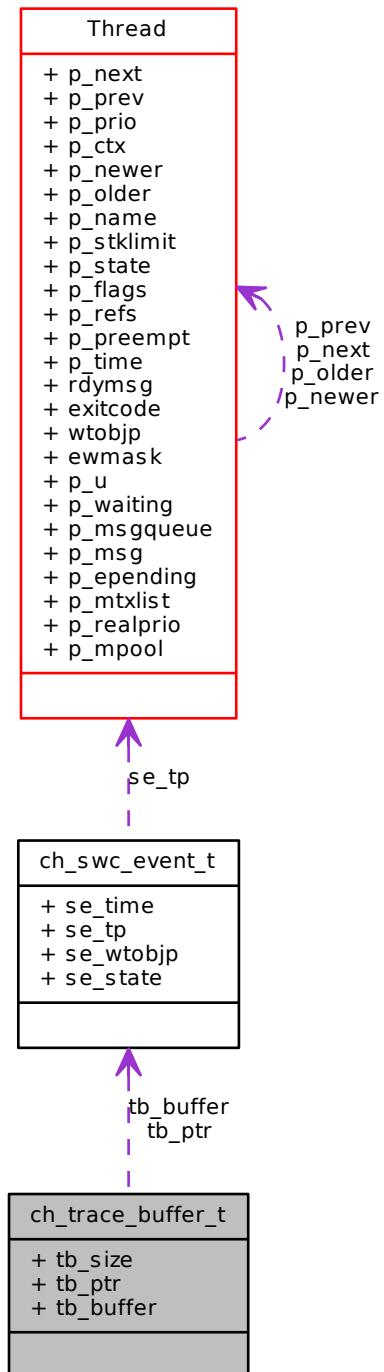
12.27 ch_trace_buffer_t Struct Reference

12.27.1 Detailed Description

Trace buffer header.

```
#include <chdebug.h>
```

Collaboration diagram for ch_trace_buffer_t:



Data Fields

- unsigned `tb_size`

Trace buffer size (entries).

- `ch_swc_event_t * tb_ptr`
Pointer to the buffer front.
- `ch_swc_event_t tb_buffer [CH_TRACE_BUFFER_SIZE]`
Ring buffer.

12.27.2 Field Documentation

12.27.2.1 unsigned ch_trace_buffer_t::tb_size

Trace buffer size (entries).

12.27.2.2 `ch_swc_event_t* ch_trace_buffer_t::tb_ptr`

Pointer to the buffer front.

12.27.2.3 `ch_swc_event_t ch_trace_buffer_t::tb_buffer[CH_TRACE_BUFFER_SIZE]`

Ring buffer.

12.28 chdebug_t Struct Reference

12.28.1 Detailed Description

ChibiOS/RT memory signature record.

```
#include <chregistry.h>
```

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 struct.
- `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`

- 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.*

12.28.2 Field Documentation

12.28.2.1 char chdebug_t::ch_identifier[4]

Always set to "main".

12.28.2.2 uint8_t chdebug_t::ch_zero

Must be zero.

12.28.2.3 uint8_t chdebug_t::ch_size

Size of this structure.

12.28.2.4 uint16_t chdebug_t::ch_version

Encoded ChibiOS/RT version.

12.28.2.5 uint8_t chdebug_t::ch_ptrsize

Size of a pointer.

12.28.2.6 uint8_t chdebug_t::ch_timesize

Size of a `systime_t`.

12.28.2.7 uint8_t chdebug_t::ch_threadsize

Size of a `Thread` struct.

12.28.2.8 uint8_t chdebug_t::cf_off_prio

Offset of `p_prio` field.

12.28.2.9 `uint8_t chdebug_t::cf_off_ctx`

Offset of `p_ctx` field.

12.28.2.10 `uint8_t chdebug_t::cf_off_newer`

Offset of `p_newer` field.

12.28.2.11 `uint8_t chdebug_t::cf_off_older`

Offset of `p_older` field.

12.28.2.12 `uint8_t chdebug_t::cf_off_name`

Offset of `p_name` field.

12.28.2.13 `uint8_t chdebug_t::cf_off_stklimit`

Offset of `p_stklimit` field.

12.28.2.14 `uint8_t chdebug_t::cf_off_state`

Offset of `p_state` field.

12.28.2.15 `uint8_t chdebug_t::cf_off_flags`

Offset of `p_flags` field.

12.28.2.16 `uint8_t chdebug_t::cf_off_refs`

Offset of `p_refs` field.

12.28.2.17 `uint8_t chdebug_t::cf_off_preempt`

Offset of `p_preempt` field.

12.28.2.18 `uint8_t chdebug_t::cf_off_time`

Offset of `p_time` field.

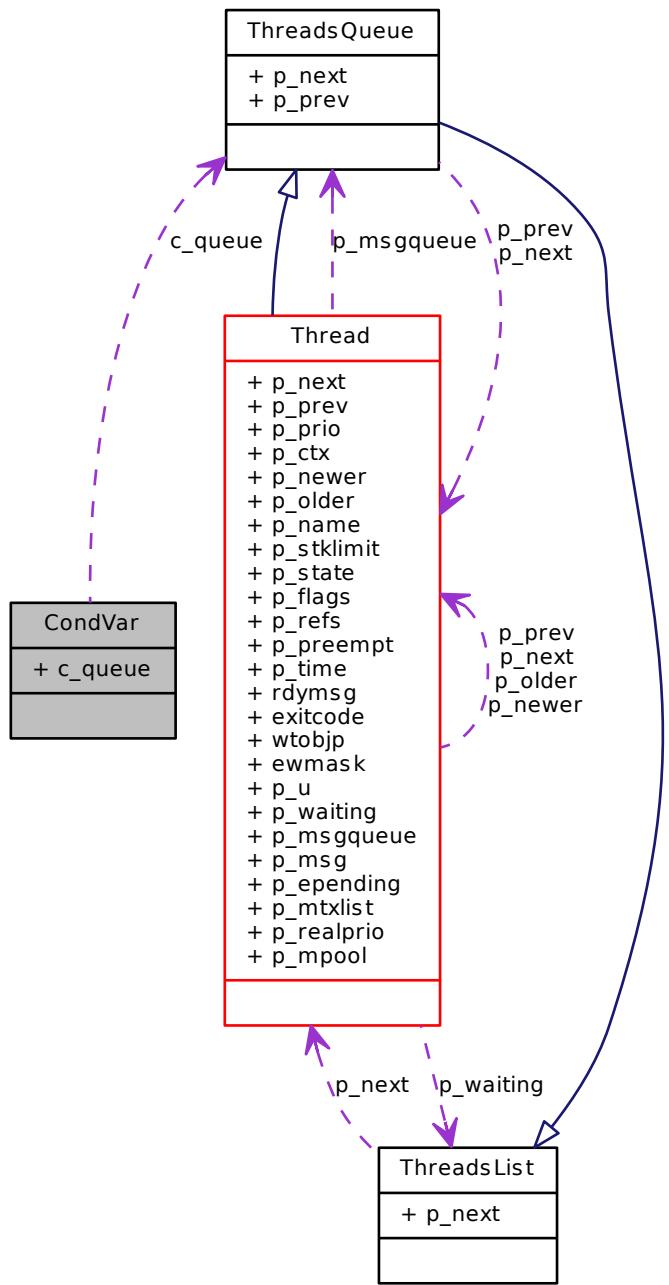
12.29 CondVar Struct Reference

12.29.1 Detailed Description

[CondVar](#) structure.

```
#include <chcond.h>
```

Collaboration diagram for CondVar:



Data Fields

- `ThreadsQueue c_queue`
CondVar threads queue.

12.29.2 Field Documentation

12.29.2.1 ThreadsQueue CondVar::c_queue

[CondVar](#) threads queue.

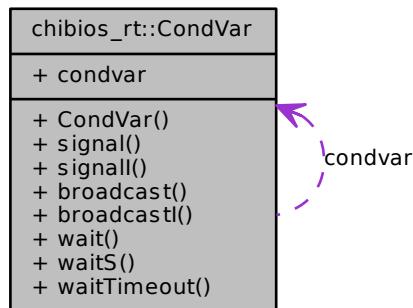
12.30 chibios_rt::CondVar Class Reference

12.30.1 Detailed Description

Class encapsulating a conditional variable.

```
#include <ch.hpp>
```

Collaboration diagram for chibios_rt::CondVar:



Public Member Functions

- [CondVar](#) (void)
CondVar object constructor.
- void [signal](#) (void)
Signals one thread that is waiting on the condition variable.
- void [signall](#) (void)
Signals one thread that is waiting on the condition variable.
- void [broadcast](#) (void)
Signals all threads that are waiting on the condition variable.
- void [broadcastl](#) (void)
Signals all threads that are waiting on the condition variable.
- [msg_t wait](#) (void)
Waits on the condition variable releasing the mutex lock.
- [msg_t waitS](#) (void)
Waits on the condition variable releasing the mutex lock.
- [msg_t waitTimeout](#) ([systime_t](#) time)
Waits on the [CondVar](#) while releasing the controlling mutex.

Data Fields

- [::CondVar condvar](#)
Embedded [CondVar](#) structure.

12.30.2 Constructor & Destructor Documentation

12.30.2.1 CondVar::CondVar (void)

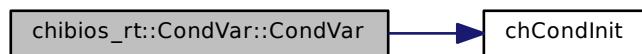
`CondVar` object constructor.

The embedded `CondVar` structure is initialized.

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:



12.30.3 Member Function Documentation

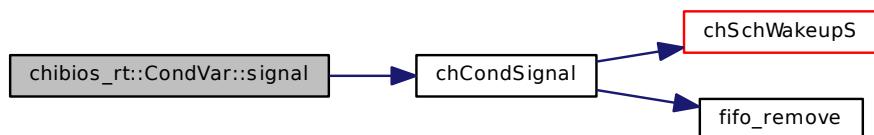
12.30.3.1 void CondVar::signal (void)

Signals one thread that is waiting on the condition variable.

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:



12.30.3.2 void CondVar::signall (void)

Signals one thread that is waiting on the condition variable.

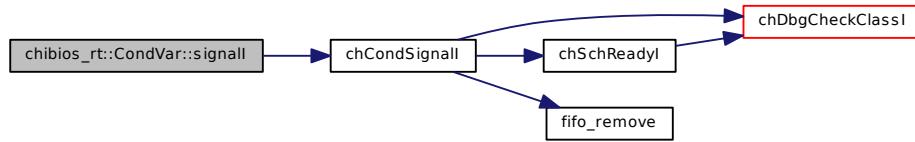
Postcondition

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.

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:

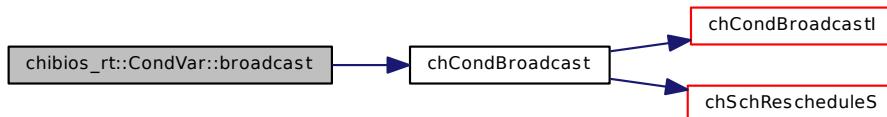
**12.30.3.3 void CondVar::broadcast(void)**

Signals all threads that are waiting on the condition variable.

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:

**12.30.3.4 void CondVar::broadcastl(void)**

Signals all threads that are waiting on the condition variable.

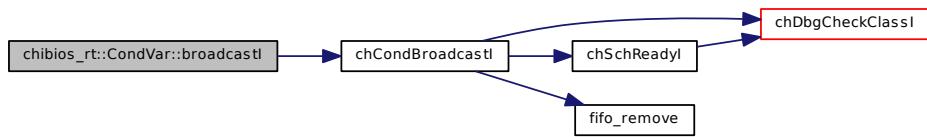
Postcondition

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.

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:



12.30.3.5 msg_t CondVar::wait(void)

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.

Returns

A message specifying how the invoking thread has been released from the condition variable.

Return values

<code>RDY_OK</code>	if the condvar has been signaled using <code>chCondSignal()</code> .
<code>RDY_RESET</code>	if the condvar 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:



12.30.3.6 msg_t CondVar::waitS(void)

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.

Returns

A message specifying how the invoking thread has been released from the condition variable.

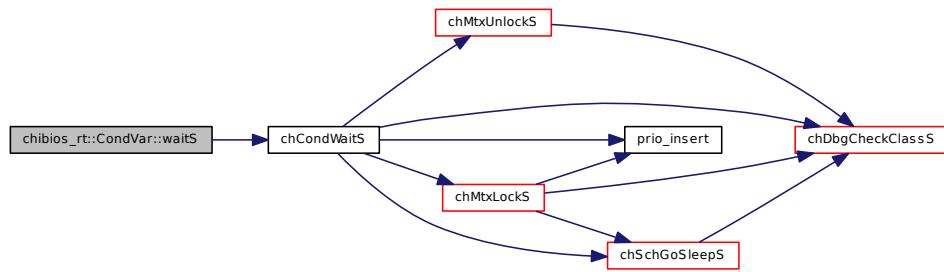
Return values

<i>RDY_OK</i>	if the condvar has been signaled using chCondSignal() .
<i>RDY_RESET</i>	if the condvar has been signaled using chCondBroadcast() .

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:

**12.30.3.7 msg_t CondVar::waitTimeout(systime_t time)**

Waits on the [CondVar](#) while releasing the controlling mutex.

Parameters

in	<i>time</i> the number of ticks before the operation fails
----	--

Returns

The wakep mode.

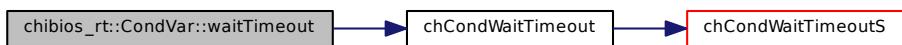
Return values

<i>RDY_OK</i>	if the condvar was signaled using chCondSignal() .
<i>RDY_RESET</i>	if the condvar was signaled using chCondBroadcast() .
<i>RDY_TIMEOUT</i>	if the condvar was not 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:



12.30.4 Field Documentation

12.30.4.1 ::CondVar chibios_rt::CondVar::condvar

Embedded [CondVar](#) structure.

12.31 context Struct Reference

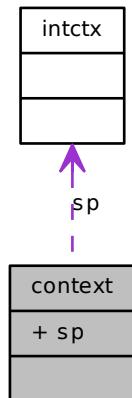
12.31.1 Detailed Description

Platform dependent part of the [Thread](#) structure.

This structure usually contains just the saved stack pointer defined as a pointer to a `intctx` structure.

```
#include <chcore.h>
```

Collaboration diagram for context:



12.32 chibios_rt::Core Class Reference

12.32.1 Detailed Description

Class encapsulating the base system functionalities.

```
#include <ch.hpp>
```

Static Public Member Functions

- static void * [alloc](#) (size_t size)
Allocates a memory block.
- static void * [alloc1](#) (size_t size)
Allocates a memory block.
- static size_t [getStatus](#) (void)
Core memory status.

12.32.2 Member Function Documentation

12.32.2.1 void * chibios_rt::Core::alloc(size_t size) [static]

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.

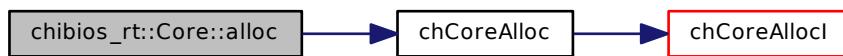
Return values

`NULL` 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:



12.32.2.2 void * chibios_rt::Core::alloc1(size_t size) [static]

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

`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:



12.32.2.3 size_t chibios_rt::Core::getStatus (void) [static]

[Core](#) memory status.

Returns

The size, in bytes, of the free core memory.

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:



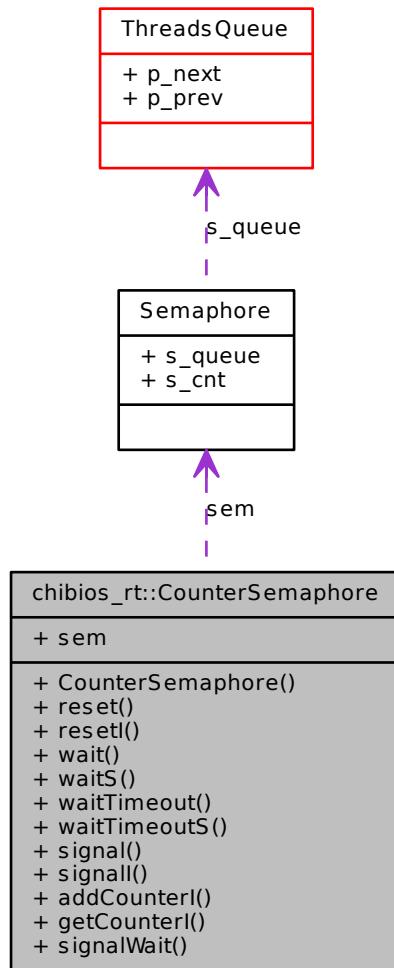
12.33 chibios_rt::CounterSemaphore Class Reference

12.33.1 Detailed Description

Class encapsulating a semaphore.

```
#include <ch.hpp>
```

Collaboration diagram for chibios_rt::CounterSemaphore:



Public Member Functions

- **CounterSemaphore (cnt_t n)**
CounterSemaphore constructor.
- **void reset (cnt_t n)**
Performs a reset operation on the semaphore.
- **void resetl (cnt_t n)**
Performs a reset operation on the semaphore.
- **msg_t wait (void)**
Performs a wait operation on a semaphore.
- **msg_t waitS (void)**
Performs a wait operation on a semaphore.
- **msg_t waitTimeout (systime_t time)**
Performs a wait operation on a semaphore with timeout specification.
- **msg_t waitTimeoutS (systime_t time)**

- `void signal (void)`
Performs a signal operation on a semaphore.
- `void signall (void)`
Performs a signal operation on a semaphore.
- `void addCounterl (cnt_t n)`
Adds the specified value to the semaphore counter.
- `cnt_t getCounterl (void)`
Returns the semaphore counter value.

Static Public Member Functions

- static `msg_t signalWait (CounterSemaphore *ssem, CounterSemaphore *wsem)`
Atomic signal and wait operations.

Data Fields

- `::Semaphore sem`
Embedded [Semaphore](#) structure.

12.33.2 Constructor & Destructor Documentation

12.33.2.1 chibios_rt::CounterSemaphore::CounterSemaphore (`cnt_t n`)

[CounterSemaphore](#) constructor.

The embedded [Semaphore](#) structure is initialized.

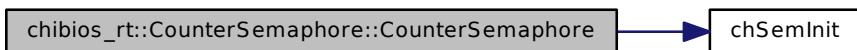
Parameters

in	<code>n</code> the semaphore counter value, must be greater or equal to zero
----	--

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:



12.33.3 Member Function Documentation

12.33.3.1 void chibios_rt::CounterSemaphore::reset (`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 waked up by a reset rather than a signal because the [chSemWait\(\)](#) will return RDY_RESET instead of RDY_OK.

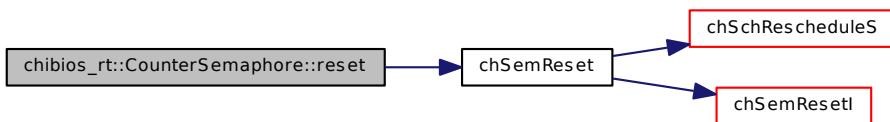
Parameters

in *n* 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:

**12.33.3.2 void chibios_rt::CounterSemaphore::resetl(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 waked up by a reset rather than a signal because the [chSemWait\(\)](#) will return RDY_RESET instead of RDY_OK.

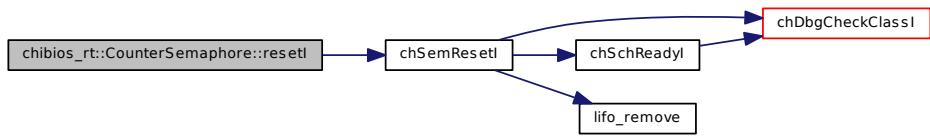
Parameters

in *n* 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:



12.33.3.3 msg_t chibios_rt::CounterSemaphore::wait(void)

Performs a wait operation on a semaphore.

Returns

A message specifying how the invoking thread has been released from the semaphore.

Return values

RDY_OK if the thread has not stopped on the semaphore or the semaphore has been signaled.

RDY_RESET if the semaphore has been reset using `chSemReset()`.

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:



12.33.3.4 msg_t chibios_rt::CounterSemaphore::waitS(void)

Performs a wait operation on a semaphore.

Returns

A message specifying how the invoking thread has been released from the semaphore.

Return values

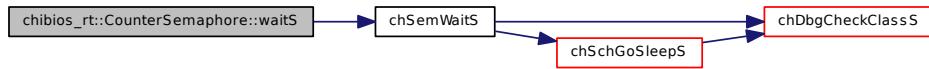
RDY_OK if the thread has not stopped on the semaphore or the semaphore has been signaled.

RDY_RESET 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:



12.33.3.5 msg_t chibios_rt::CounterSemaphore::waitTimeout (systime_t time)

Performs a wait operation on a semaphore with timeout specification.

Parameters

- in *time* the number of ticks before the operation timeouts, the following special values are allowed:
- *TIME_IMMEDIATE* immediate timeout.
 - *TIME_INFINITE* no timeout.

Returns

A message specifying how the invoking thread has been released from the semaphore.

Return values

- | | |
|--------------------|--|
| <i>RDY_OK</i> | if the thread has not stopped on the semaphore or the semaphore has been signaled. |
| <i>RDY_RESET</i> | if the semaphore has been reset using chSemReset () . |
| <i>RDY_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:



12.33.3.6 msg_t chibios_rt::CounterSemaphore::waitTimeoutS (systime_t time)

Performs a wait operation on a semaphore with timeout specification.

Parameters

- in *time* the number of ticks before the operation timeouts, the following special values are allowed:
- *TIME_IMMEDIATE* immediate timeout.
 - *TIME_INFINITE* no timeout.

Returns

A message specifying how the invoking thread has been released from the semaphore.

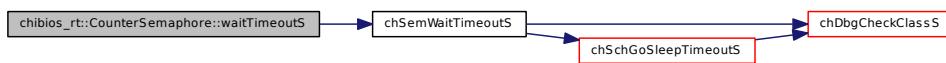
Return values

- RDY_OK* if the thread has not stopped on the semaphore or the semaphore has been signaled.
- RDY_RESET* if the semaphore has been reset using [chSemReset \(\)](#).
- RDY_TIMEOUT* 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:

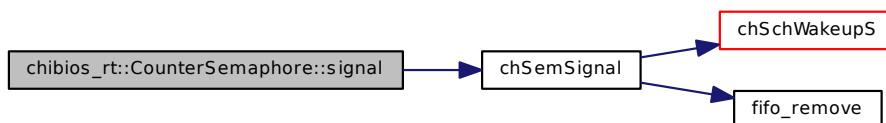
**12.33.3.7 void chibios_rt::CounterSemaphore::signal (void)**

Performs a signal operation on a semaphore.

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:

**12.33.3.8 void chibios_rt::CounterSemaphore::signall (void)**

Performs a signal operation on a semaphore.

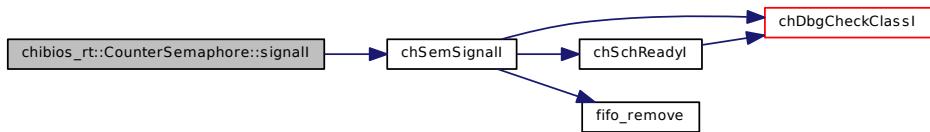
Postcondition

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.

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:



12.33.3.9 void chibios_rt::CounterSemaphore::addCounterl (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.

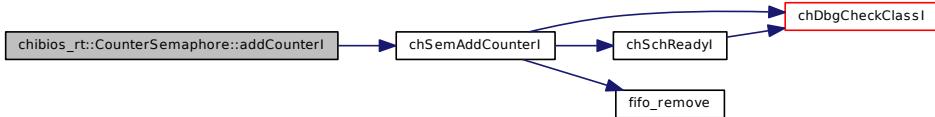
Parameters

in *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:



12.33.3.10 cnt_t chibios_rt::CounterSemaphore::getCounterl (void)

Returns the semaphore counter value.

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.

12.33.3.11 `msg_t chibios_rt::CounterSemaphore::signalWait (CounterSemaphore * ssem, CounterSemaphore * wsem) [static]`

Atomic signal and wait operations.

Parameters

in	<code>ssem</code> Semaphore object to be signaled
in	<code>wsem</code> Semaphore object to wait on

Returns

A message specifying how the invoking thread has been released from the semaphore.

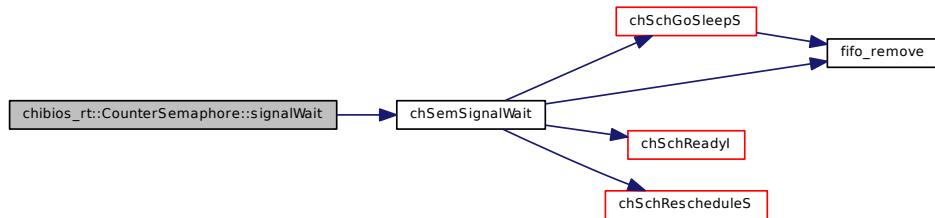
Return values

<code>RDY_OK</code>	if the thread has not stopped on the semaphore or the semaphore has been signaled.
<code>RDY_RESET</code>	if the semaphore has been reset using chSemReset () .

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:



12.33.4 Field Documentation

12.33.4.1 ::Semaphore chibios_rt::CounterSemaphore::sem

Embedded [Semaphore](#) structure.

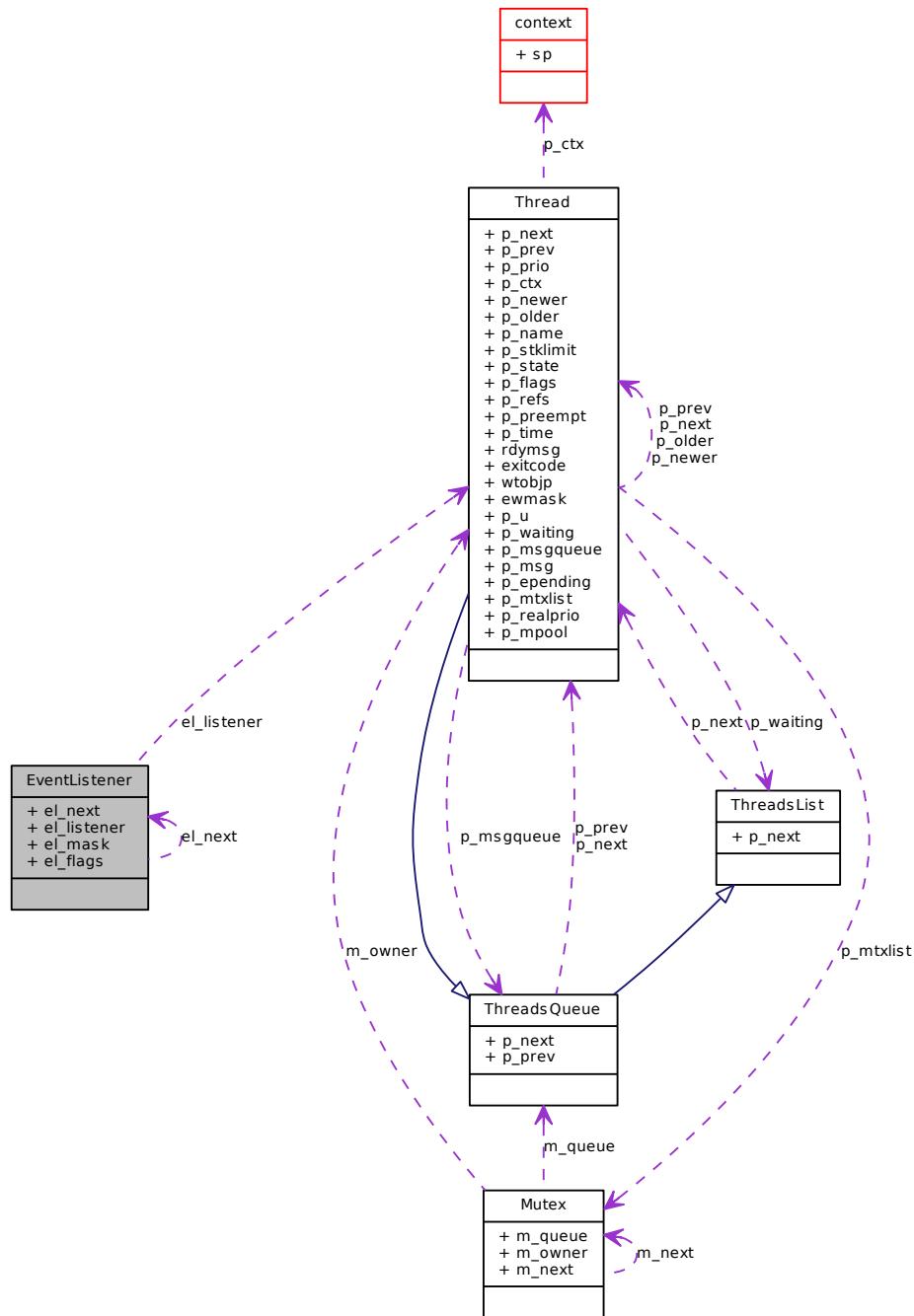
12.34 EventListener Struct Reference

12.34.1 Detailed Description

Event Listener structure.

```
#include <chevents.h>
```

Collaboration diagram for EventListener:



Data Fields

- `EventListener * el_next`
Next Event Listener registered on the Event Source.
 - `Thread * el_listener`
Thread interested in the Event Source.
 - `eventmask_t el_mask`

Event flags mask associated by the thread to the Event Source.

- `flagsmask_t el_flags`

Flags added to the listener by the event source.

12.34.2 Field Documentation

12.34.2.1 `EventListener* EventListener::el_next`

Next Event Listener registered on the Event Source.

12.34.2.2 `Thread* EventListener::el_listener`

`Thread` interested in the Event Source.

12.34.2.3 `eventmask_t EventListener::el_mask`

Event flags mask associated by the thread to the Event Source.

12.34.2.4 `flagsmask_t EventListener::el_flags`

Flags added to the listener by the event source.

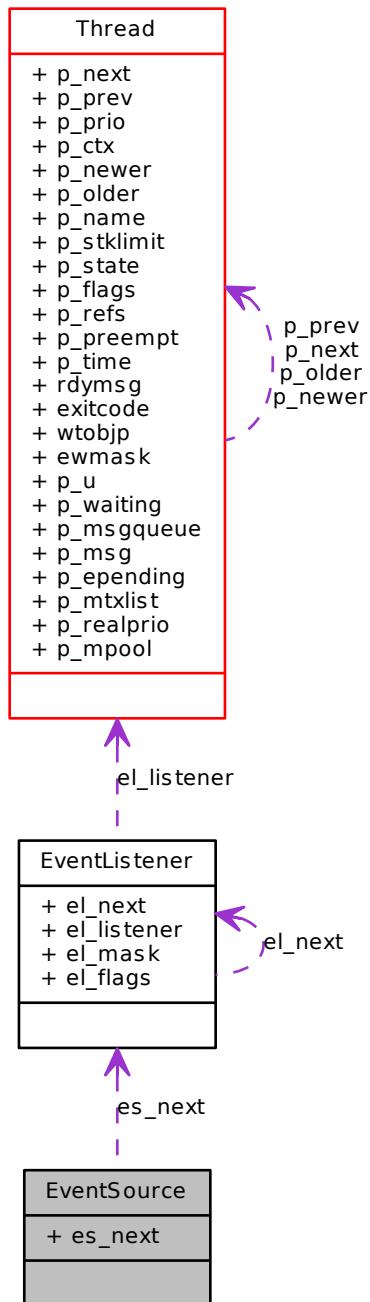
12.35 EventSource Struct Reference

12.35.1 Detailed Description

Event Source structure.

```
#include <chevents.h>
```

Collaboration diagram for EventSource:



Data Fields

- `EventListener * es_next`

First Event Listener registered on the Event Source.

12.35.2 Field Documentation

12.35.2.1 EventListener* EventSource::es_next

First Event Listener registered on the Event Source.

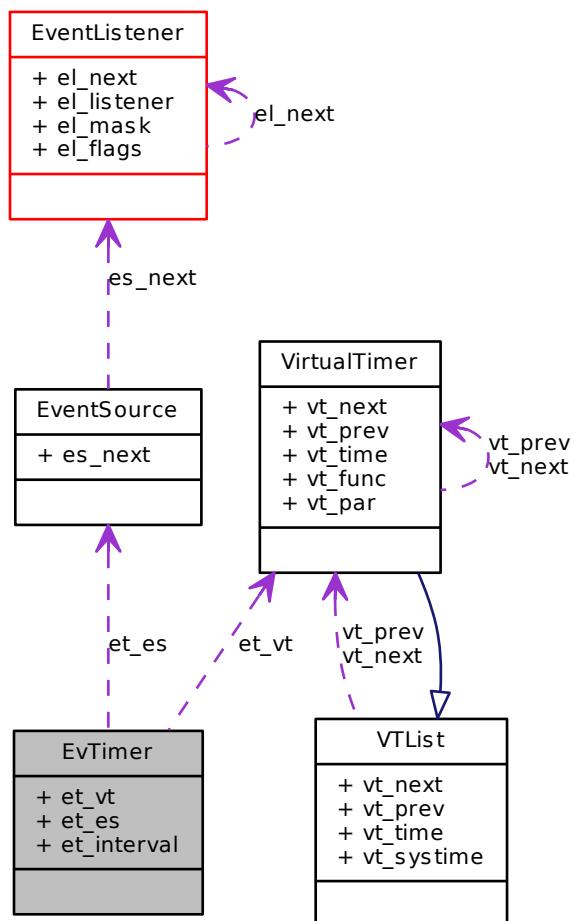
12.36 EvTimer Struct Reference

12.36.1 Detailed Description

Event timer structure.

```
#include <evtimer.h>
```

Collaboration diagram for EvTimer:



12.37 chibios_rt::EvtListener Class Reference

12.37.1 Detailed Description

Class encapsulating an event listener.

```
#include <ch.hpp>
```

Public Member Functions

- flagsmask_t [getAndClearFlags](#) (void)
Returns the pending flags from the listener and clears them.
- flagsmask_t [getAndClearFlagsI](#) (void)
Returns the flags associated to an [EventListener](#).

Data Fields

- struct::EventListener [ev_listener](#)
Embedded [Event Listener](#) structure.

12.37.2 Member Function Documentation

12.37.2.1 flagsmask_t chibios_rt::EvtListener::getAndClearFlags(void)

Returns the pending flags from the listener and clears them.

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:



12.37.2.2 flagsmask_t chibios_rt::EvtListener::getAndClearFlagsI(void)

Returns the flags associated to an [EventListener](#).

The flags are returned and the [EventListener](#) flags mask is cleared.

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.

Here is the call graph for this function:



12.37.3 Field Documentation

12.37.3.1 struct ::EventListener chibios_rt::EvtListener::ev_listener

Embedded [EventListener](#) structure.

12.38 chibios_rt::EvtSource Class Reference

12.38.1 Detailed Description

Class encapsulating an event source.

```
#include <ch.hpp>
```

Public Member Functions

- [EvtSource \(void\)](#)
EvtSource object constructor.
- [void registerOne \(chibios_rt::EvtListener *elp, eventid_t eid\)](#)
Registers a listener on the event source.
- [void registerMask \(chibios_rt::EvtListener *elp, eventmask_t emask\)](#)
Registers an Event Listener on an Event Source.
- [void unregister \(chibios_rt::EvtListener *elp\)](#)
Unregisters a listener.
- [void broadcastFlags \(flagsmask_t flags\)](#)
Broadcasts on an event source.
- [void broadcastFlags1 \(flagsmask_t flags\)](#)
Broadcasts on an event source.

Data Fields

- [struct::EventSource ev_source](#)
Embedded [EventSource](#) structure.

12.38.2 Constructor & Destructor Documentation

12.38.2.1 chibios_rt::EvtSource::EvtSource (void)

[EvtSource](#) object constructor.

The embedded [EventSource](#) structure is initialized.

Function Class:

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

12.38.3 Member Function Documentation**12.38.3.1 void chibios_rt::EvtSource::registerOne (chibios_rt::EvtListener * elp, eventid_t eid)**

Registers a listener on the event source.

Parameters

in	<i>elp</i>	pointer to the EvtListener object
in	<i>eid</i>	numeric identifier assigned to the Event Listener

Function Class:

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

12.38.3.2 void chibios_rt::EvtSource::registerMask (chibios_rt::EvtListener * elp, eventmask_t emask)

Registers an Event Listener on an Event Source.

Note

Multiple Event Listeners can specify the same bits to be added.

Parameters

in	<i>elp</i>	pointer to the EvtListener object
in	<i>emask</i>	the mask of event flags to be pended 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:

**12.38.3.3 void chibios_rt::EvtSource::unregister (chibios_rt::EvtListener * elp)**

Unregisters a listener.

The specified listeners is no more signaled by the event source.

Parameters

in	<i>elp</i>	the listener to be unregistered
----	------------	---------------------------------

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:

**12.38.3.4 void chibios_rt::EvtSource::broadcastFlags (flagsmask_t flags)**

Broadcasts on an event source.

All the listeners registered on the event source are signaled and the flags are added to the listener's flags mask.

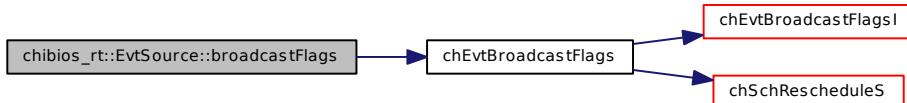
Parameters

in *flags* 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:

**12.38.3.5 void chibios_rt::EvtSource::broadcastFlags1 (flagsmask_t flags)**

Broadcasts on an event source.

All the listeners registered on the event source are signaled and the flags are added to the listener's flags mask.

Parameters

in *flags* 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:



12.38.4 Field Documentation

12.38.4.1 struct ::EventSource chibios_rt::EvtSource::ev_source

Embedded [EventSource](#) structure.

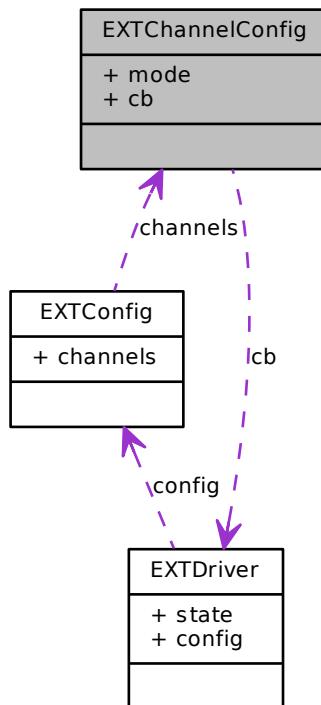
12.39 EXTChannelConfig Struct Reference

12.39.1 Detailed Description

Channel configuration structure.

```
#include <ext_lld.h>
```

Collaboration diagram for EXTChannelConfig:



Data Fields

- `uint32_t mode`
Channel mode.
- `extcallback_t cb`
Channel callback.

12.39.2 Field Documentation

12.39.2.1 `uint32_t EXTChannelConfig::mode`

Channel mode.

12.39.2.2 `extcallback_t EXTChannelConfig::cb`

Channel callback.

In the STM32 implementation a `NULL` callback pointer is valid and configures the channel as an event sources instead of an interrupt source.

12.40 EXTConfig Struct Reference

12.40.1 Detailed Description

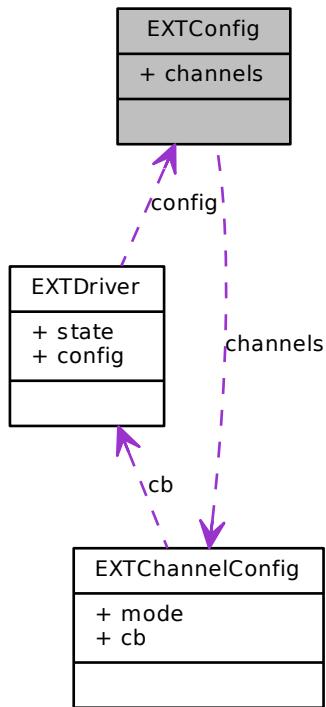
Driver configuration structure.

Note

It could be empty on some architectures.

```
#include <ext_llld.h>
```

Collaboration diagram for EXTConfig:



Data Fields

- `EXTChannelConfig channels [EXT_MAX_CHANNELS]`

Channel configurations.

12.40.2 Field Documentation

12.40.2.1 EXTChannelConfig EXTConfig::channels[EXT_MAX_CHANNELS]

Channel configurations.

12.41 extctx Struct Reference

12.41.1 Detailed Description

Interrupt saved context.

This structure represents the stack frame saved during a preemption-capable interrupt handler.

```
#include <chcore.h>
```

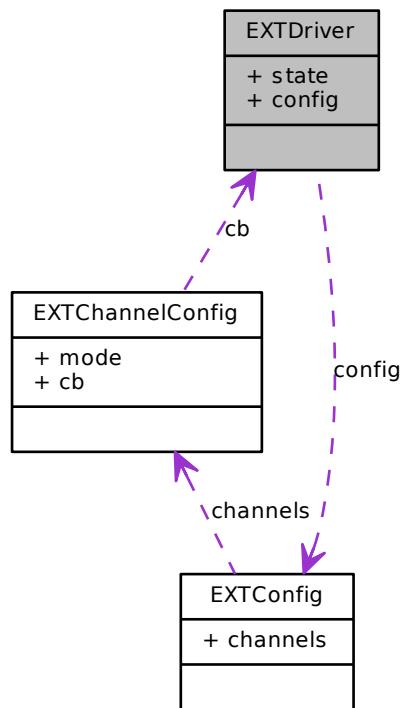
12.42 EXTDriver Struct Reference

12.42.1 Detailed Description

Structure representing an EXT driver.

```
#include <ext_lld.h>
```

Collaboration diagram for EXTDriver:



Data Fields

- **extstate_t state**
Driver state.
- **const EXTConfig * config**
Current configuration data.

12.42.2 Field Documentation

12.42.2.1 extstate_t EXTDriver::state

Driver state.

12.42.2.2 const EXTConfig* EXTDriver::config

Current configuration data.

12.43 GenericQueue Struct Reference

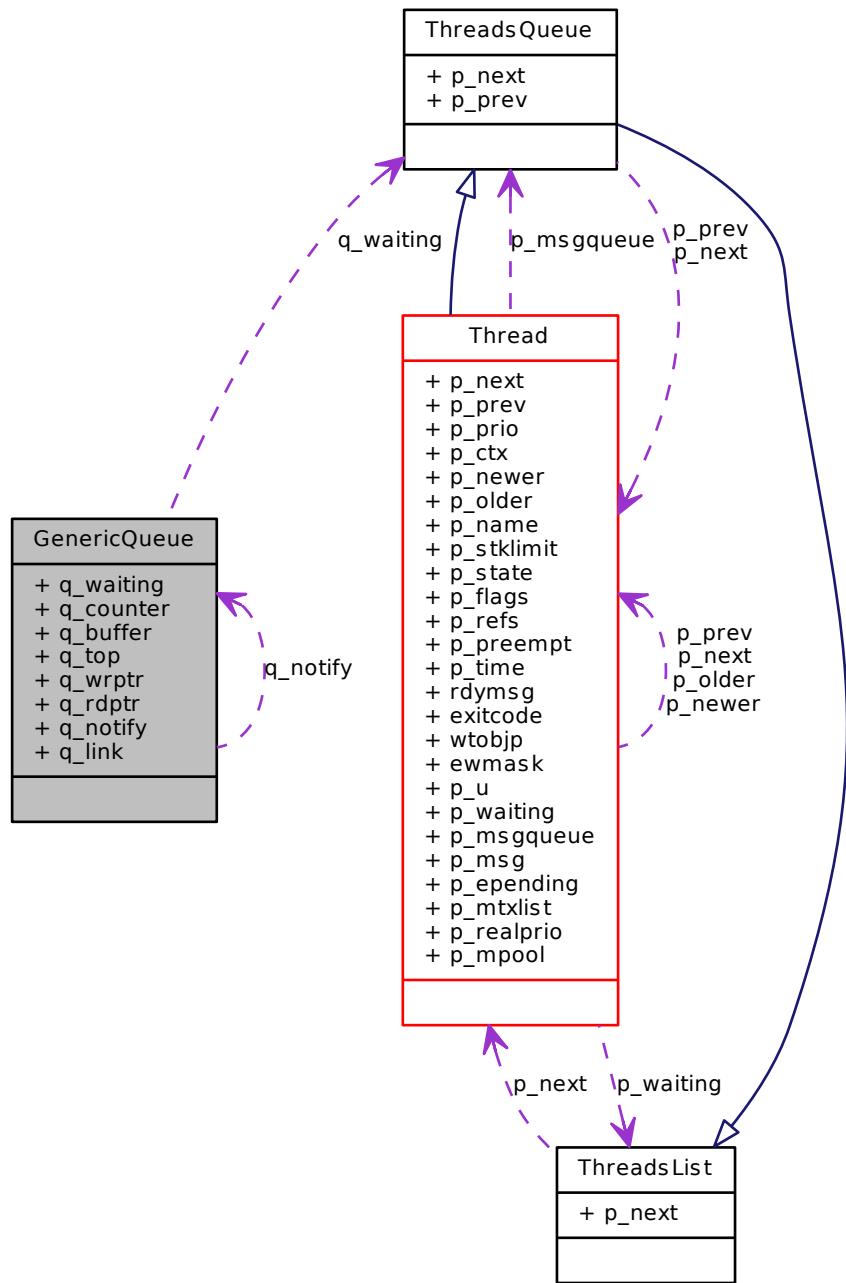
12.43.1 Detailed Description

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.

```
#include <chqueues.h>
```

Collaboration diagram for GenericQueue:



Data Fields

- **ThreadsQueue q_waiting**
Queue of waiting threads.
- **size_t q_counter**
Resources counter.
- **uint8_t * q_buffer**

- `uint8_t * q_top`
Pointer to the queue buffer.
- `uint8_t * q_wptr`
Pointer to the first location after the buffer.
- `uint8_t * q_rptr`
Write pointer.
- `uint8_t * q_link`
Read pointer.
- `qnotify_t q_notify`
Data notification callback.
- `void * q_link`
Application defined field.

12.43.2 Field Documentation

12.43.2.1 ThreadsQueue GenericQueue::q_waiting

Queue of waiting threads.

12.43.2.2 size_t GenericQueue::q_counter

Resources counter.

12.43.2.3 uint8_t* GenericQueue::q_buffer

Pointer to the queue buffer.

12.43.2.4 uint8_t* GenericQueue::q_top

Pointer to the first location after the buffer.

12.43.2.5 uint8_t* GenericQueue::q_wptr

Write pointer.

12.43.2.6 uint8_t* GenericQueue::q_rptr

Read pointer.

12.43.2.7 qnotify_t GenericQueue::q_notify

Data notification callback.

12.43.2.8 void* GenericQueue::q_link

Application defined field.

12.44 GPTConfig Struct Reference

12.44.1 Detailed Description

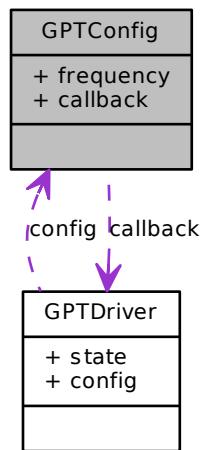
Driver configuration structure.

Note

It could be empty on some architectures.

```
#include <gpt_ll.h>
```

Collaboration diagram for GPTConfig:



Data Fields

- `gptfreq_t frequency`
Timer clock in Hz.
- `gptcallback_t callback`
Timer callback pointer.

12.44.2 Field Documentation

12.44.2.1 `gptfreq_t GPTConfig::frequency`

Timer clock in Hz.

Note

The low level can use assertions in order to catch invalid frequency specifications.

12.44.2.2 `gptcallback_t` GPTConfig::callback

Timer callback pointer.

Note

This callback is invoked on GPT counter events.

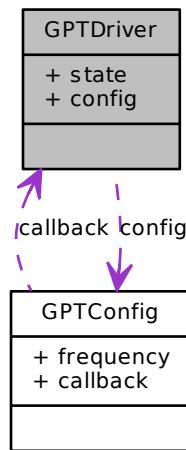
12.45 GPTDriver Struct Reference

12.45.1 Detailed Description

Structure representing a GPT driver.

```
#include <gpt_lld.h>
```

Collaboration diagram for GPTDriver:



Data Fields

- `gptstate_t state`
Driver state.
- `const GPTConfig * config`
Current configuration data.

12.45.2 Field Documentation

12.45.2.1 `gptstate_t` GPTDriver::state

Driver state.

12.45.2.2 const GPTConfig* GPTDriver::config

Current configuration data.

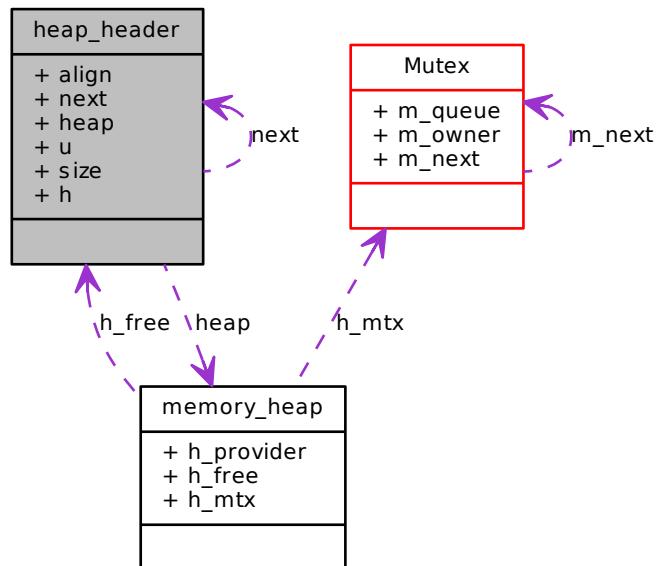
12.46 heap_header Union Reference

12.46.1 Detailed Description

Memory heap block header.

```
#include <chheap.h>
```

Collaboration diagram for heap_header:



12.46.2 Field Documentation

12.46.2.1 union heap_header* heap_header::next

Next block in free list.

12.46.2.2 MemoryHeap* heap_header::heap

Block owner heap.

12.46.2.3 union { ... } heap_header::u

Overlapped fields.

12.46.2.4 `size_t heap_header::size`

Size of the memory block.

12.47 I2CConfig Struct Reference

12.47.1 Detailed Description

Driver configuration structure.

Note

Implementations may extend this structure to contain more, architecture dependent, fields. Driver configuration structure.

```
#include <i2c_lld.h>
```

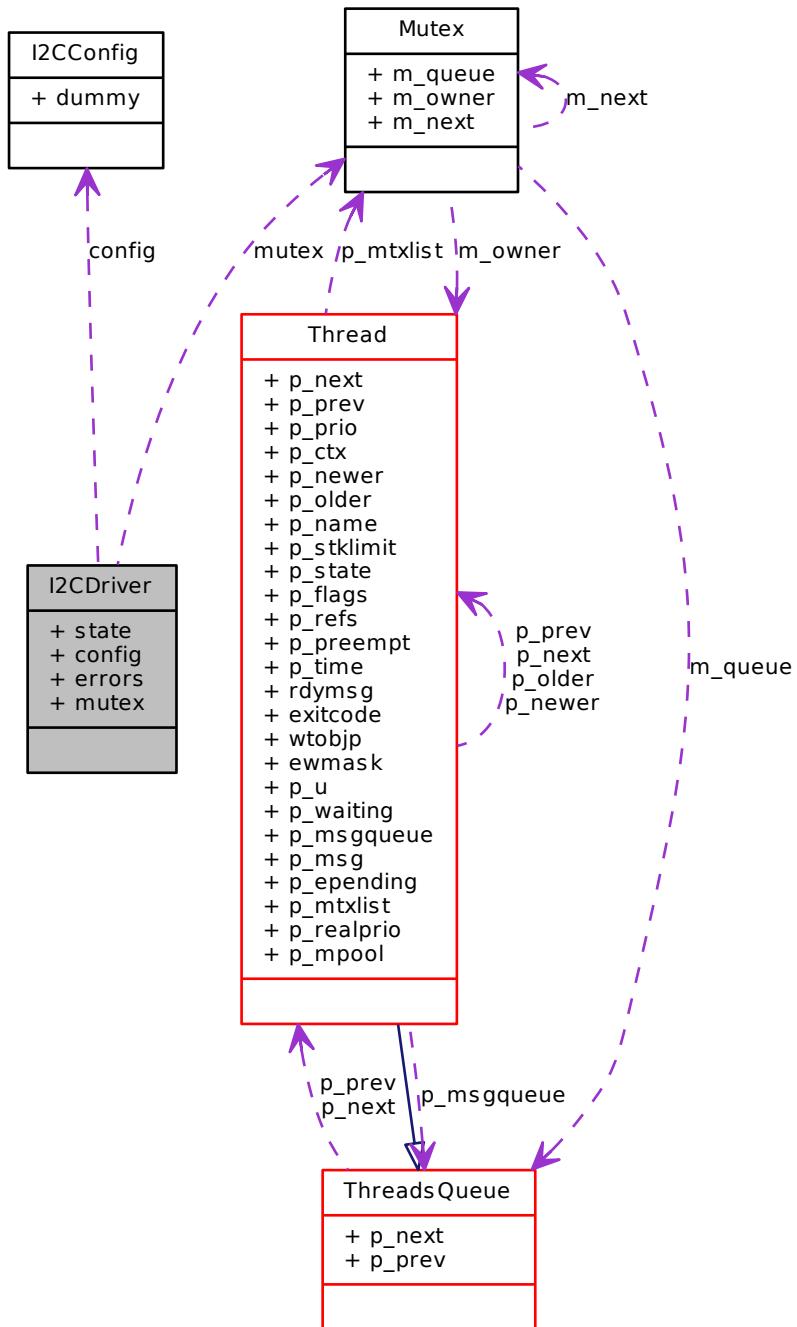
12.48 I2CDriver Struct Reference

12.48.1 Detailed Description

Structure representing an I2C driver.

```
#include <i2c_lld.h>
```

Collaboration diagram for I2CDriver:



Data Fields

- `i2cstate_t state`
Driver state.
- `const I2CConfig * config`

- *i2cflags_t errors*
Error flags.
- *Mutex mutex*
Mutex protecting the bus.

12.48.2 Field Documentation

12.48.2.1 i2cstate_t I2CDriver::state

Driver state.

12.48.2.2 const I2CConfig* I2CDriver::config

Current configuration data.

12.48.2.3 i2cflags_t I2CDriver::errors

Error flags.

12.48.2.4 Mutex I2CDriver::mutex

Mutex protecting the bus.

12.49 ICUConfig Struct Reference

12.49.1 Detailed Description

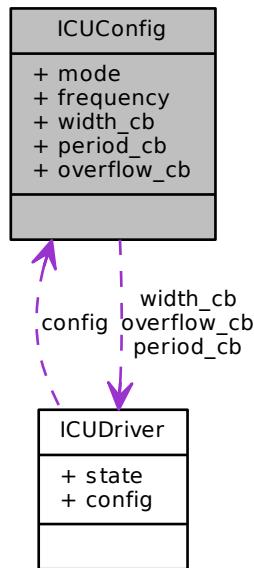
Driver configuration structure.

Note

It could be empty on some architectures.

```
#include <icu_lld.h>
```

Collaboration diagram for ICUConfig:



Data Fields

- `icemode_t mode`
Driver mode.
- `icufreq_t frequency`
Timer clock in Hz.
- `icucallback_t width_cb`
Callback for pulse width measurement.
- `icucallback_t period_cb`
Callback for cycle period measurement.
- `icucallback_t overflow_cb`
Callback for timer overflow.

12.49.2 Field Documentation

12.49.2.1 `icemode_t ICUConfig::mode`

Driver mode.

12.49.2.2 `icufreq_t ICUConfig::frequency`

Timer clock in Hz.

Note

The low level can use assertions in order to catch invalid frequency specifications.

12.49.2.3 `icucallback_t ICUConfig::width_cb`

Callback for pulse width measurement.

12.49.2.4 `icucallback_t ICUConfig::period_cb`

Callback for cycle period measurement.

12.49.2.5 `icucallback_t ICUConfig::overflow_cb`

Callback for timer overflow.

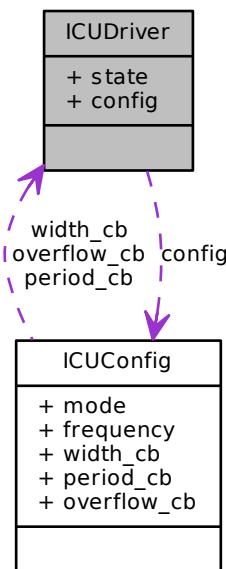
12.50 ICUDriver Struct Reference

12.50.1 Detailed Description

Structure representing an ICU driver.

```
#include <icu_ll.h>
```

Collaboration diagram for ICUDriver:



Data Fields

- `icustate_t state`
Driver state.
- `const ICUConfig * config`
Current configuration data.

12.50.2 Field Documentation

12.50.2.1 icustate_t ICUDriver::state

Driver state.

12.50.2.2 const ICUConfig* ICUDriver::config

Current configuration data.

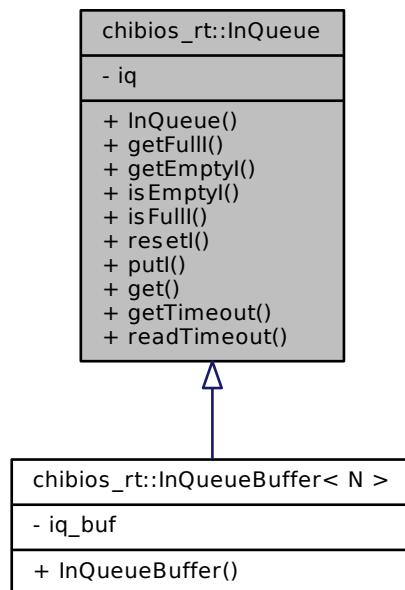
12.51 chibios_rt::InQueue Class Reference

12.51.1 Detailed Description

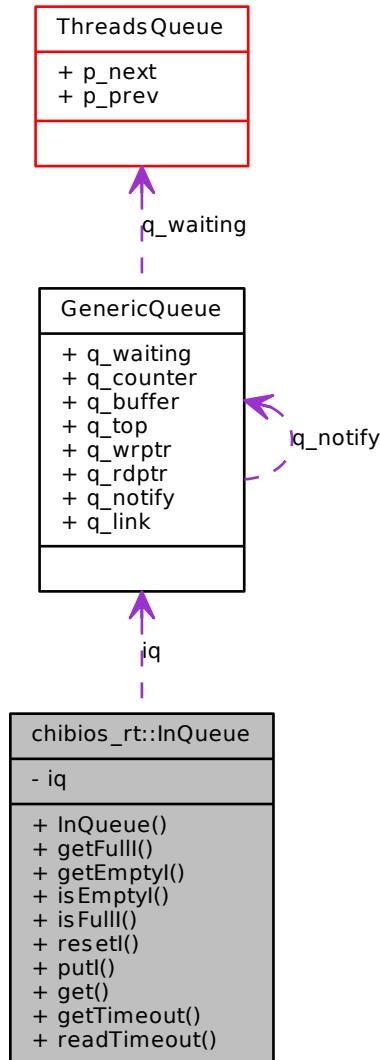
Class encapsulating an input queue.

```
#include <ch.hpp>
```

Inheritance diagram for chibios_rt::InQueue:



Collaboration diagram for chibios_rt::InQueue:



Public Member Functions

- `InQueue (uint8_t *bp, size_t size, qnotify_t infy, void *link)`
InQueue constructor.
- `size_t getFulll ()`
Returns the filled space into an input queue.
- `size_t getEmptyl ()`
Returns the empty space into an input queue.
- `bool isEmptyl ()`
Evaluates to TRUE if the specified input queue is empty.
- `bool isFulll ()`
Evaluates to TRUE if the specified input queue is full.

- void [resetl](#) (void)
Resets an input queue.
- [msg_t putl](#) (uint8_t b)
Input queue write.
- [msg_t get](#) ()
Input queue read.
- [msg_t getTimeout](#) (systime_t time)
Input queue read with timeout.
- size_t [readTimeout](#) (uint8_t *bp, size_t n, systime_t time)
Input queue read with timeout.

12.51.2 Constructor & Destructor Documentation

12.51.2.1 chibios_rt::InQueue::InQueue (uint8_t * bp, size_t size, qnotify_t infy, void * link)

[InQueue](#) constructor.

Parameters

in	<i>bp</i>	pointer to a memory area allocated as queue buffer
in	<i>size</i>	size of the queue buffer
in	<i>infy</i>	pointer to a callback function that is invoked when data is read from the queue. The value can be NULL.
in	<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:



12.51.3 Member Function Documentation

12.51.3.1 size_t chibios_rt::InQueue::getFull (void)

Returns the filled space into an input queue.

Returns

The number of full bytes in the queue.

Return values

0 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.

12.51.3.2 size_t chibios_rt::InQueue::getEmpty()

Returns the empty space into an input queue.

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.

12.51.3.3 bool chibios_rt::InQueue::isEmpty()

Evaluates to TRUE if the specified input queue is empty.

Returns

The queue status.

Return values

false if the queue is not empty.
true 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.

12.51.3.4 bool chibios_rt::InQueue::isFull()

Evaluates to TRUE if the specified input queue is full.

Returns

The queue status.

Return values

FALSE if the queue is not full.
TRUE 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.

12.51.3.5 void chibios_rt::InQueue::resetl (void)

Resets an input queue.

All the data in the input queue is erased and lost, any waiting thread is resumed with status Q_RESET.

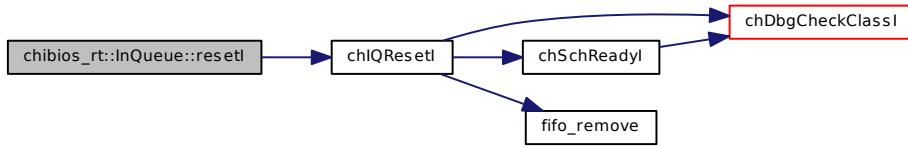
Note

A reset operation can be used by a low level driver in order to obtain immediate attention from the high level layers.

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:



12.51.3.6 msg_t chibios_rt::InQueue::putl (uint8_t b)

Input queue write.

A byte value is written into the low end of an input queue.

Parameters

in	<i>b</i> the byte value to be written in the queue
----	--

Returns

The operation status.

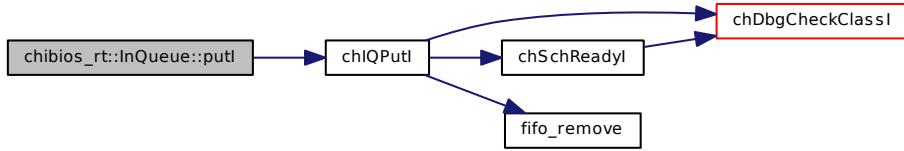
Return values

<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:



12.51.3.7 `msg_t chibios_rt::InQueue::get()`

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.

Returns

A byte value from the queue.

Return values

`Q_RESET` 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.

12.51.3.8 `msg_t chibios_rt::InQueue::getTimeout(systime_t time)`

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 `THD_STATE_WTQUEUE`.

Parameters

in	<code>time</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

A byte value from the queue.

Return values

`Q_TIMEOUT` if the specified time expired.

`Q_RESET` 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:



12.51.3.9 size_t chibios_rt::InQueue::readTimeout (uint8_t * bp, size_t n, systime_t time)

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 THD_STATE_WTQUEUE.

Parameters

<code>out</code>	<code>bp</code>	pointer to the data buffer
<code>in</code>	<code>n</code>	the maximum amount of data to be transferred, the value 0 is reserved
<code>in</code>	<code>time</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.

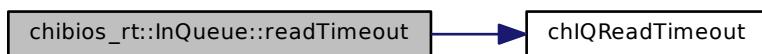
Returns

The 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:



12.52 chibios_rt::InQueueBuffer< N > Class Template Reference

12.52.1 Detailed Description

```
template<int N>class chibios_rt::InQueueBuffer< N >
```

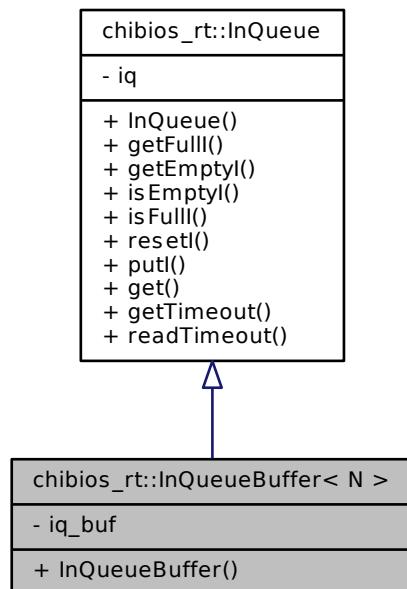
Template class encapsulating an input queue and its buffer.

Parameters

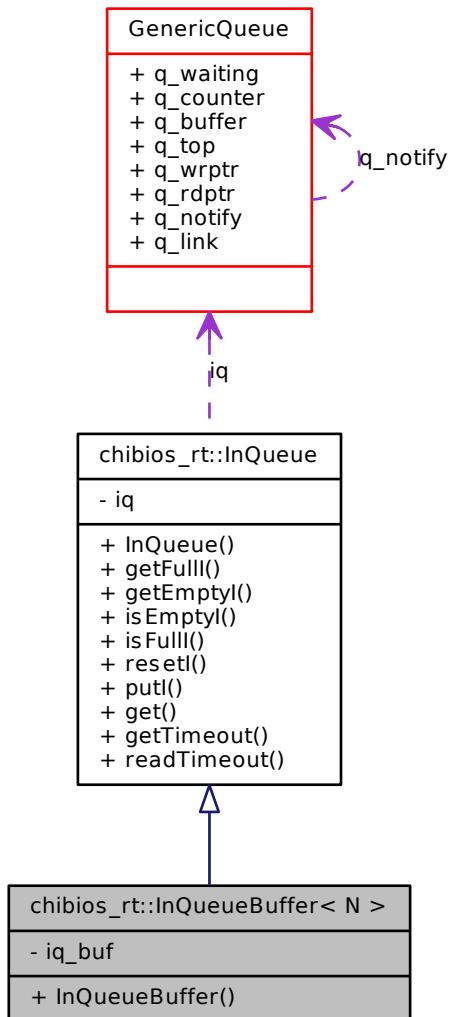
N size of the input queue

```
#include <ch.hpp>
```

Inheritance diagram for chibios_rt::InQueueBuffer< N >:



Collaboration diagram for chibios_rt::InQueueBuffer< N >:



Public Member Functions

- `InQueueBuffer (qnotify_t infy, void *link)`
`InQueueBuffer` constructor.

12.52.2 Constructor & Destructor Documentation

12.52.2.1 template<int N> `chibios_rt::InQueueBuffer< N >::InQueueBuffer (qnotify_t infy, void * link)`
`[inline]`

`InQueueBuffer` constructor.

Parameters

in	<code>infy</code>	input notify callback function
in	<code>link</code>	parameter to be passed to the callback

Function Class:

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

12.53 intctx Struct Reference

12.53.1 Detailed Description

System saved context.

This structure represents the inner stack frame during a context switching.

```
#include <chcore.h>
```

12.54 IOBus Struct Reference

12.54.1 Detailed Description

I/O bus descriptor.

This structure describes a group of contiguous digital I/O lines that have to be handled as bus.

Note

I/O operations on a bus do not affect I/O lines on the same port but not belonging to the bus.

```
#include <pal.h>
```

Data Fields

- **ioportid_t portid**
Port identifier.
- **ioportmask_t mask**
Bus mask aligned to port bit 0.
- **uint_fast8_t offset**
Offset, within the port, of the least significant bit of the bus.

12.54.2 Field Documentation

12.54.2.1 ioportid_t IOBus::portid

Port identifier.

12.54.2.2 ioportmask_t IOBus::mask

Bus mask aligned to port bit 0.

Note

The bus mask implicitly define the bus width. A logical AND is performed on the bus data.

12.54.2.3 uint_fast8_t IOBus::offset

Offset, within the port, of the least significant bit of the bus.

12.55 MACConfig Struct Reference

12.55.1 Detailed Description

Driver configuration structure.

```
#include <mac_lld.h>
```

Data Fields

- `uint8_t * mac_address`

MAC address.

12.55.2 Field Documentation

12.55.2.1 `uint8_t* MACConfig::mac_address`

MAC address.

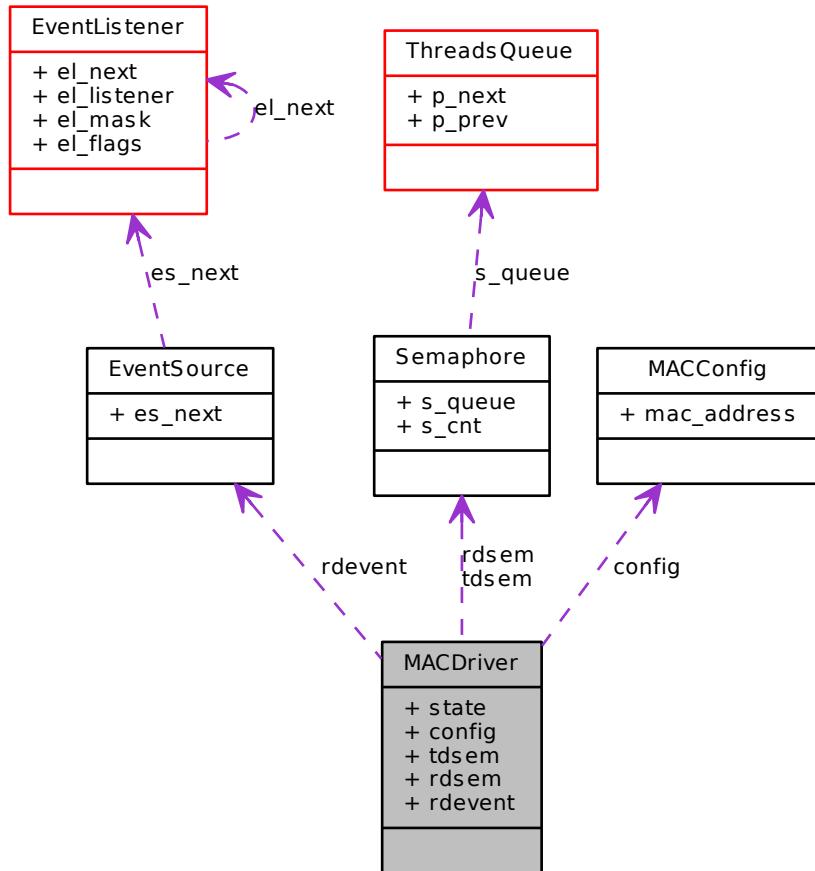
12.56 MACDriver Struct Reference

12.56.1 Detailed Description

Structure representing a MAC driver.

```
#include <mac_lld.h>
```

Collaboration diagram for MACDriver:



Data Fields

- **macstate_t state**
Driver state.
- **const MACConfig * config**
Current configuration data.
- **Semaphore tdsem**
Transmit semaphore.
- **Semaphore rdsem**
Receive semaphore.
- **EventSource rdevent**
Receive event.

12.56.2 Field Documentation

12.56.2.1 macstate_t MACDriver::state

Driver state.

12.56.2.2 const MACConfig* MACDriver::config

Current configuration data.

12.56.2.3 Semaphore MACDriver::tdsem

Transmit semaphore.

12.56.2.4 Semaphore MACDriver::rdsem

Receive semaphore.

12.56.2.5 EventSource MACDriver::rdevent

Receive event.

12.57 MACReceiveDescriptor Struct Reference

12.57.1 Detailed Description

Structure representing a receive descriptor.

```
#include <mac_lld.h>
```

Data Fields

- size_t **offset**

Current read offset.

- size_t **size**

Available data size.

12.57.2 Field Documentation

12.57.2.1 size_t MACReceiveDescriptor::offset

Current read offset.

12.57.2.2 size_t MACReceiveDescriptor::size

Available data size.

12.58 MACTransmitDescriptor Struct Reference

12.58.1 Detailed Description

Structure representing a transmit descriptor.

```
#include <mac_lld.h>
```

Data Fields

- size_t **offset**
Current write offset.
- size_t **size**
Available space size.

12.58.2 Field Documentation

12.58.2.1 size_t MACTransmitDescriptor::offset

Current write offset.

12.58.2.2 size_t MACTransmitDescriptor::size

Available space size.

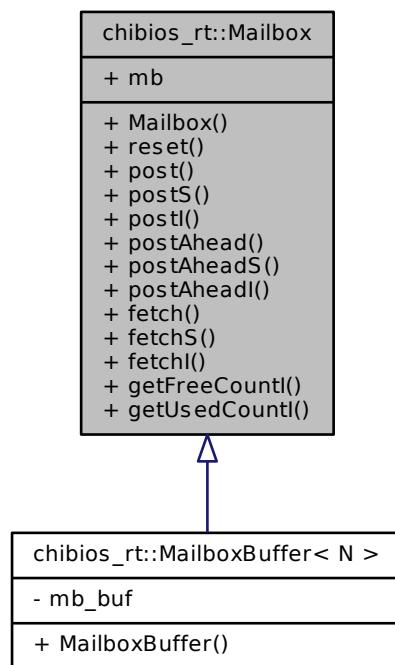
12.59 chibios_rt::Mailbox Class Reference

12.59.1 Detailed Description

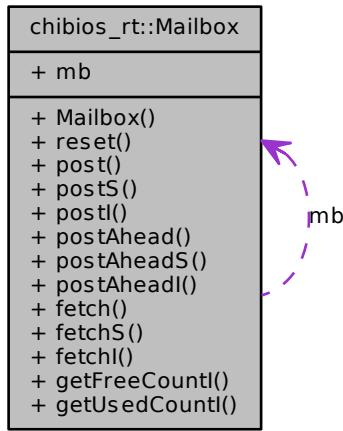
Class encapsulating a mailbox.

```
#include <ch.hpp>
```

Inheritance diagram for chibios_rt::Mailbox:



Collaboration diagram for chibios_rt::Mailbox:



Public Member Functions

- **Mailbox (msg_t *buf, cnt_t n)**
Mailbox constructor.
- **void reset (void)**
Resets a Mailbox object.
- **msg_t post (msg_t msg, systime_t time)**
Posts a message into a mailbox.
- **msg_t postS (msg_t msg, systime_t time)**
Posts a message into a mailbox.
- **msg_t postI (msg_t msg)**
Posts a message into a mailbox.
- **msg_t postAhead (msg_t msg, systime_t time)**
Posts an high priority message into a mailbox.
- **msg_t postAheadS (msg_t msg, systime_t time)**
Posts an high priority message into a mailbox.
- **msg_t postAheadI (msg_t msg)**
Posts an high priority message into a mailbox.
- **msg_t fetch (msg_t *msgp, systime_t time)**
Retrieves a message from a mailbox.
- **msg_t fetchS (msg_t *msgp, systime_t time)**
Retrieves a message from a mailbox.
- **msg_t fetchI (msg_t *msgp)**
Retrieves a message from a mailbox.
- **cnt_t getFreeCountI (void)**
Returns the number of free message slots into a mailbox.
- **cnt_t getUsedCountI (void)**
Returns the number of used message slots into a mailbox.

Data Fields

- `::Mailbox mb`
Embedded Mailbox structure.

12.59.2 Constructor & Destructor Documentation

12.59.2.1 Mailbox::Mailbox (`msg_t *buf, cnt_t n`)

`Mailbox` constructor.

The embedded `Mailbox` structure is initialized.

Parameters

<code>in</code>	<code>buf</code>	pointer to the messages buffer as an array of <code>msg_t</code>
	<code>n</code>	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:



12.59.3 Member Function Documentation

12.59.3.1 void Mailbox::reset (void)

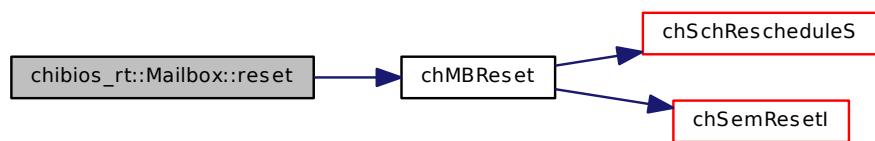
Resets a `Mailbox` object.

All the waiting threads are resumed with status `RDY_RESET` and the queued messages are lost.

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:



12.59.3.2 `msg_t Mailbox::post(msg_t msg, systime_t time)`

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

<code>in</code>	<code>msg</code>	the message to be posted on the mailbox
<code>in</code>	<code>time</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.

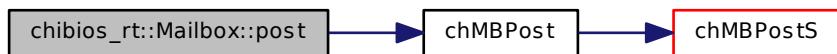
Return values

<code>RDY_OK</code>	if a message has been correctly posted.
<code>RDY_RESET</code>	if the mailbox has been reset while waiting.
<code>RDY_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:



12.59.3.3 `msg_t Mailbox::postS(msg_t msg, systime_t time)`

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

<code>in</code>	<code>msg</code>	the message to be posted on the mailbox
<code>in</code>	<code>time</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.

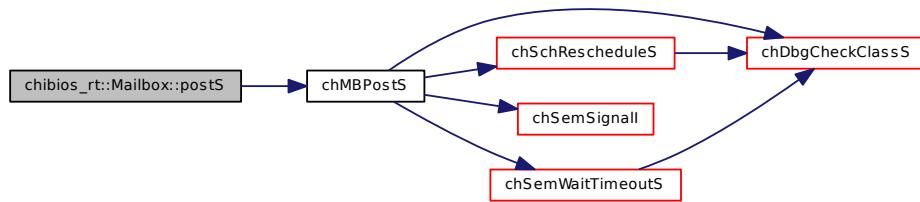
Return values

RDY_OK if a message has been correctly posted.
RDY_RESET if the mailbox has been reset while waiting.
RDY_TIMEOUT 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:

**12.59.3.4 msg_t Mailbox::postl (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 *msg* the message to be posted on the mailbox

Returns

The operation status.

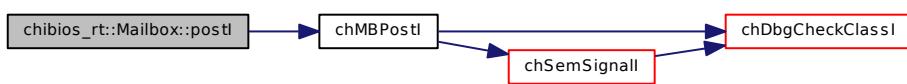
Return values

RDY_OK if a message has been correctly posted.
RDY_TIMEOUT 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:



12.59.3.5 `msg_t Mailbox::postAhead (msg_t msg, systime_t time)`

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

<code>in</code>	<code>msg</code>	the message to be posted on the mailbox
<code>in</code>	<code>time</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.

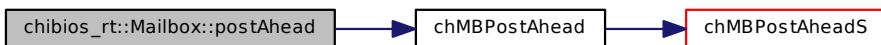
Return values

<code>RDY_OK</code>	if a message has been correctly posted.
<code>RDY_RESET</code>	if the mailbox has been reset while waiting.
<code>RDY_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:



12.59.3.6 `msg_t Mailbox::postAheadS (msg_t msg, systime_t time)`

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

<code>in</code>	<code>msg</code>	the message to be posted on the mailbox
<code>in</code>	<code>time</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.

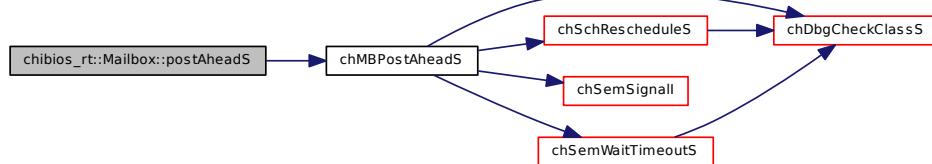
Return values

RDY_OK if a message has been correctly posted.
RDY_RESET if the mailbox has been reset while waiting.
RDY_TIMEOUT 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:

**12.59.3.7 msg_t Mailbox::postAheadl (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 *msg* the message to be posted on the mailbox

Returns

The operation status.

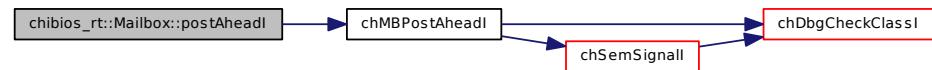
Return values

RDY_OK if a message has been correctly posted.
RDY_TIMEOUT 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:



12.59.3.8 `msg_t Mailbox::fetch(msg_t * msgp, systime_t time)`

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

<code>out</code>	<code>msgp</code>	pointer to a message variable for the received
<code>in</code>	<code>time</code>	message 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>RDY_OK</code>	if a message has been correctly fetched.
<code>RDY_RESET</code>	if the mailbox has been reset while waiting.
<code>RDY_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:



12.59.3.9 `msg_t Mailbox::fetchS(msg_t * msgp, systime_t time)`

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

<code>out</code>	<code>msgp</code>	pointer to a message variable for the received
<code>in</code>	<code>time</code>	message 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.

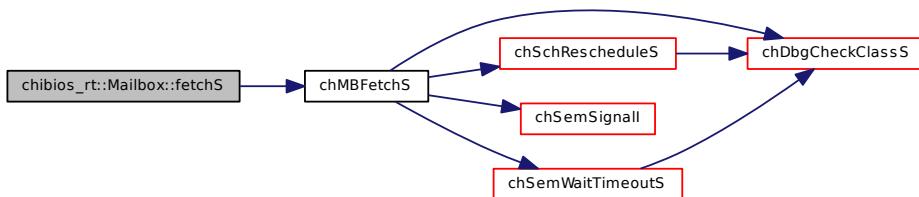
Return values

RDY_OK if a message has been correctly fetched.
RDY_RESET if the mailbox has been reset while waiting.
RDY_TIMEOUT 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:

**12.59.3.10 msg_t Mailbox::fetchl (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

out *msgp* pointer to a message variable for the received message

Returns

The operation status.

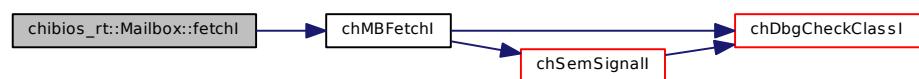
Return values

RDY_OK if a message has been correctly fetched.
RDY_TIMEOUT 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:



12.59.3.11 cnt_t Mailbox::getFreeCount(void)

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.

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.

12.59.3.12 cnt_t Mailbox::getUsedCount(void)

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.

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.

12.59.4 Field Documentation**12.59.4.1 ::Mailbox chibios_rt::Mailbox::mb**

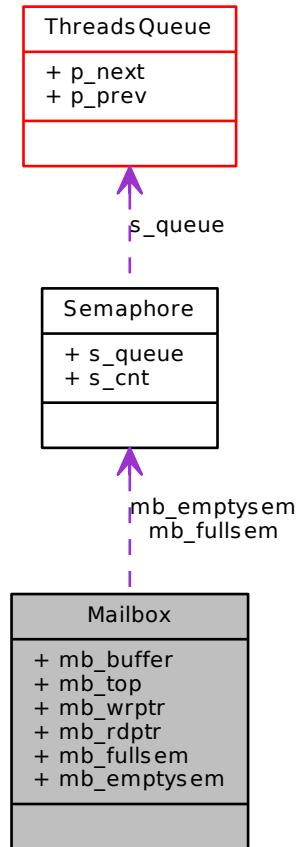
Embedded [Mailbox](#) structure.

12.60 Mailbox Struct Reference**12.60.1 Detailed Description**

Structure representing a mailbox object.

```
#include <chmboxes.h>
```

Collaboration diagram for Mailbox:



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_wptr`
Write pointer.
- `msg_t * mb_rptr`
Read pointer.
- `Semaphore mb_fullsem`
Full counter [Semaphore](#).
- `Semaphore mb_emptysem`
Empty counter [Semaphore](#).

12.60.2 Field Documentation

12.60.2.1 msg_t* Mailbox::mb_buffer

Pointer to the mailbox buffer.

12.60.2.2 msg_t* Mailbox::mb_top

Pointer to the location after the buffer.

12.60.2.3 msg_t* Mailbox::mb_wptr

Write pointer.

12.60.2.4 msg_t* Mailbox::mb_rptr

Read pointer.

12.60.2.5 Semaphore Mailbox::mb_fullsem

Full counter [Semaphore](#).

12.60.2.6 Semaphore Mailbox::mb_emptysem

Empty counter [Semaphore](#).

12.61 chibios_rt::MailboxBuffer< N > Class Template Reference

12.61.1 Detailed Description

```
template<int N>class chibios_rt::MailboxBuffer< N >
```

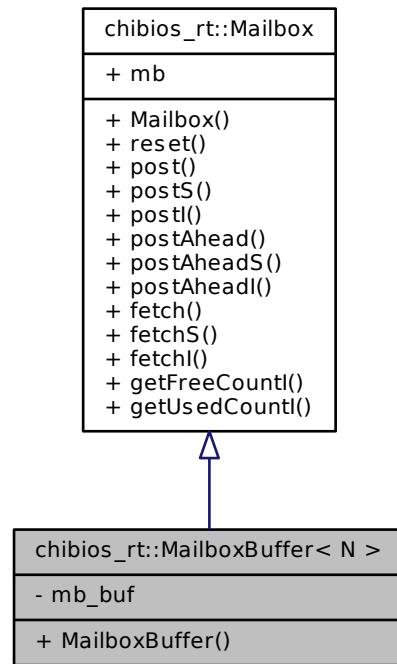
Template class encapsulating a mailbox and its messages buffer.

Parameters

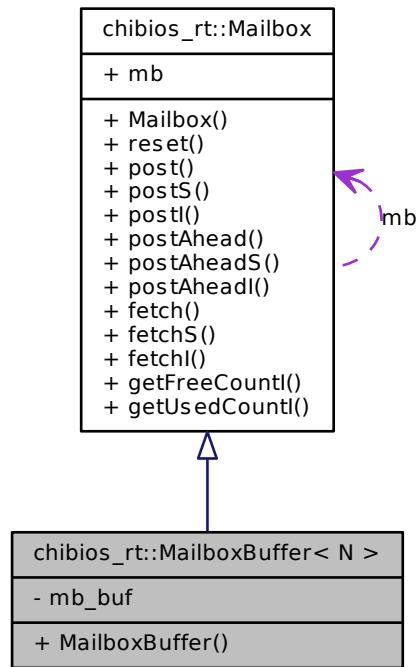
N size of the mailbox

```
#include <ch.hpp>
```

Inheritance diagram for chibios_rt::MailboxBuffer< N >:



Collaboration diagram for chibios_rt::MailboxBuffer< N >:



Public Member Functions

- [MailboxBuffer \(void\)](#)
BufferMailbox constructor.

12.61.2 Constructor & Destructor Documentation

12.61.2.1 template<int N> **chibios_rt::MailboxBuffer< N >::MailboxBuffer (void)** [inline]

BufferMailbox constructor.

Function Class:

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

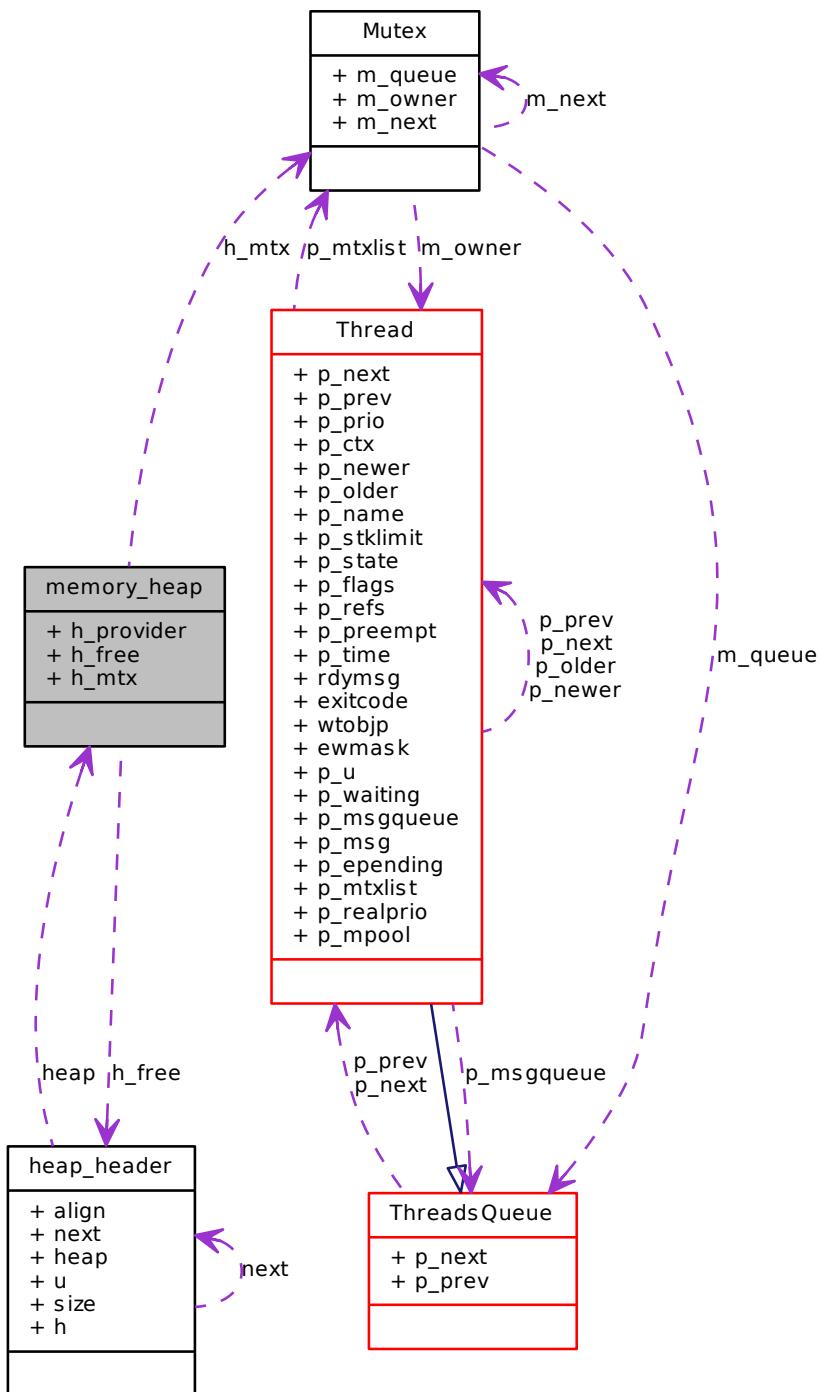
12.62 memory_heap Struct Reference

12.62.1 Detailed Description

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 h_mtx`
Heap access mutex.

12.62.2 Field Documentation

12.62.2.1 `memgetfunc_t memory_heap::h_provider`

Memory blocks provider for this heap.

12.62.2.2 `union heap_header memory_heap::h_free`

Free blocks list header.

12.62.2.3 `Mutex memory_heap::h_mtx`

Heap access mutex.

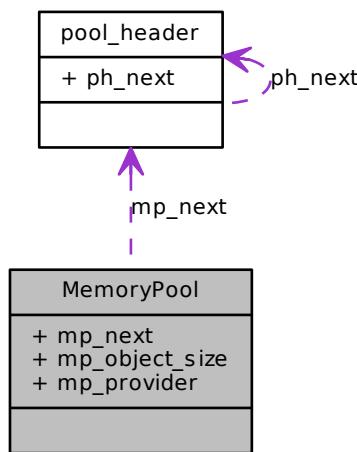
12.63 MemoryPool Struct Reference

12.63.1 Detailed Description

Memory pool descriptor.

```
#include <chmempools.h>
```

Collaboration diagram for MemoryPool:



Data Fields

- struct `pool_header * mp_next`

- `size_t mp_object_size`
Memory pool objects size.
- `memgetfunc_t mp_provider`
Memory blocks provider for this pool.

12.63.2 Field Documentation

12.63.2.1 `struct pool_header* MemoryPool::mp_next`

Pointer to the header.

12.63.2.2 `size_t MemoryPool::mp_object_size`

Memory pool objects size.

12.63.2.3 `memgetfunc_t MemoryPool::mp_provider`

Memory blocks provider for this pool.

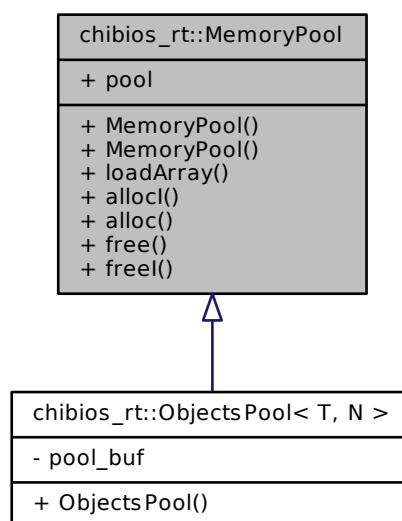
12.64 chibios_rt::MemoryPool Class Reference

12.64.1 Detailed Description

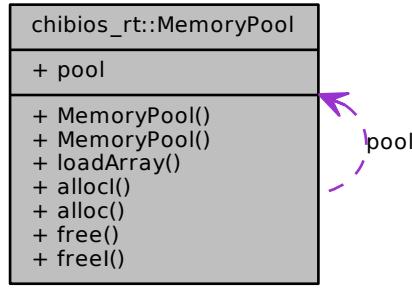
Class encapsulating a mailbox.

```
#include <ch.hpp>
```

Inheritance diagram for chibios_rt::MemoryPool:



Collaboration diagram for chibios_rt::MemoryPool:



Public Member Functions

- **MemoryPool** (size_t size, memgetfunc_t provider)
MemoryPool constructor.
- **MemoryPool** (size_t size, memgetfunc_t provider, void *p, size_t n)
MemoryPool constructor.
- void **loadArray** (void *p, size_t n)
Loads a memory pool with an array of static objects.
- void * **alloc1** (void)
Allocates an object from a memory pool.
- void * **alloc** (void)
Allocates an object from a memory pool.
- void **free** (void *objp)
Releases an object into a memory pool.
- void **freel** (void *objp)
Adds an object to a memory pool.

Data Fields

- ::MemoryPool pool
Embedded MemoryPool structure.

12.64.2 Constructor & Destructor Documentation

12.64.2.1 MemoryPool::MemoryPool (size_t size, memgetfunc_t provider)

MemoryPool constructor.

Parameters

in	size	the size of the objects contained in this memory pool, the minimum accepted size is the size of a pointer to void.
in	provider	memory provider function for the memory pool or NULL 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.

Here is the call graph for this function:

**12.64.2.2 MemoryPool::MemoryPool (size_t size, memgetfunc_t provider, void * p, size_t n)**

[MemoryPool](#) constructor.

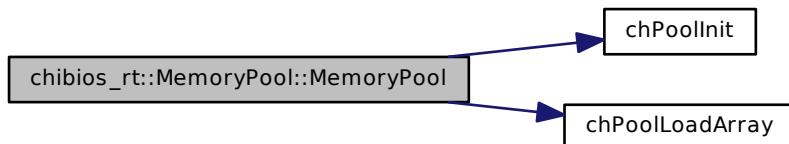
Parameters

in	size	the size of the objects contained in this memory pool, the minimum accepted size is the size of a pointer to void.
in	provider	memory provider function for the memory pool or <code>NULL</code> if the pool is not allowed to grow automatically
in	p	pointer to the array first element
in	n	number of elements in the 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:

**12.64.3 Member Function Documentation****12.64.3.1 void MemoryPool::loadArray (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>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:

**12.64.3.2 void * MemoryPool::alloc(void)**

Allocates an object from a memory pool.

Precondition

The memory pool must be already been initialized.

Returns

The pointer to the allocated object.

Return values

NULL 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:



12.64.3.3 void * MemoryPool::alloc (void)

Allocates an object from a memory pool.

Precondition

The memory pool must be already been initialized.

Returns

The pointer to the allocated object.

Return values

NULL 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:



12.64.3.4 void MemoryPool::free (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.

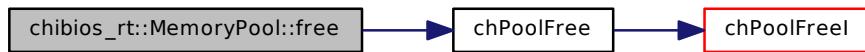
Parameters

in *objp* 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:



12.64.3.5 void MemoryPool::freel (void * *objp*)

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` *objp* 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:



12.64.4 Field Documentation

12.64.4.1 ::MemoryPool chibios_rt::MemoryPool::pool

Embedded `MemoryPool` structure.

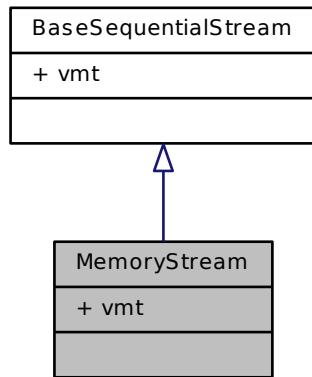
12.65 MemoryStream Struct Reference

12.65.1 Detailed Description

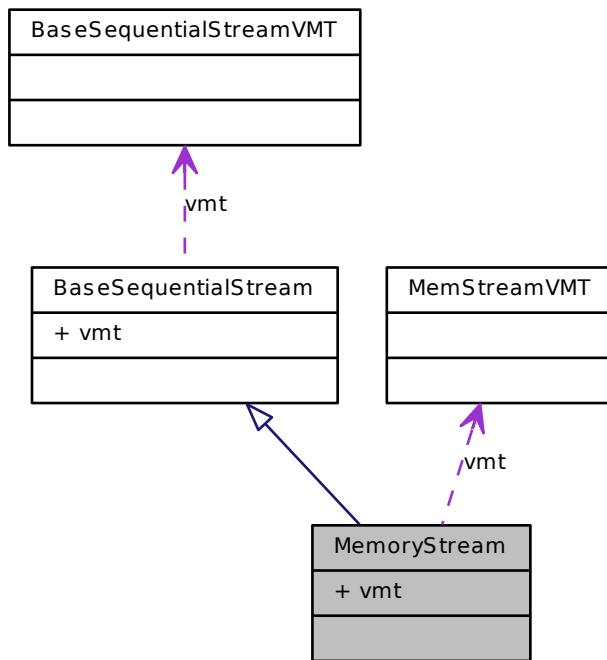
Memory stream object.

```
#include <memstreams.h>
```

Inheritance diagram for MemoryStream:



Collaboration diagram for MemoryStream:



Data Fields

- struct [MemStreamVMT](#) * vmt
Virtual Methods Table.

12.65.2 Field Documentation

12.65.2.1 struct MemStreamVMT* MemoryStream::vmt

Virtual Methods Table.

Reimplemented from [BaseSequentialStream](#).

12.66 MemStreamVMT Struct Reference

12.66.1 Detailed Description

MemStream virtual methods table.

```
#include <memstreams.h>
```

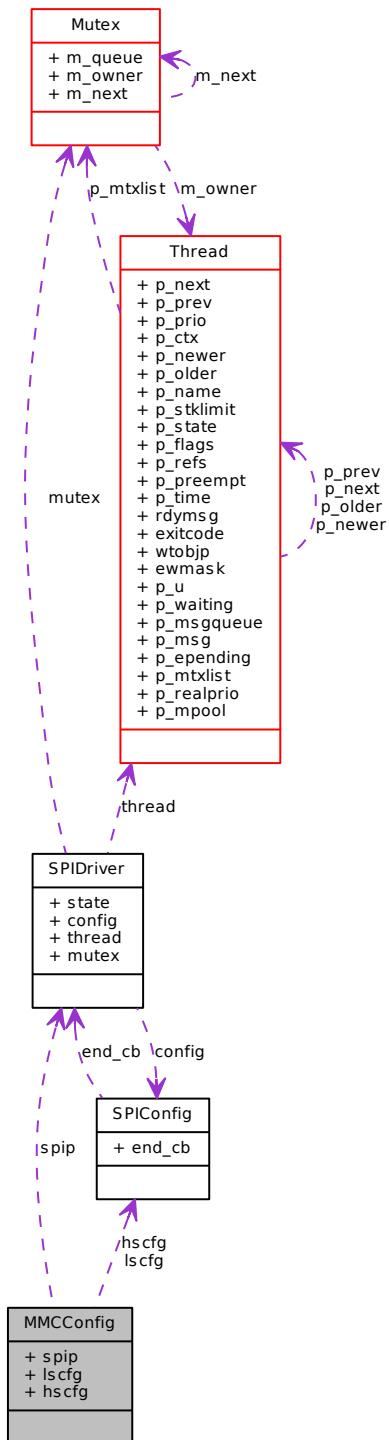
12.67 MMCConfig Struct Reference

12.67.1 Detailed Description

MMC/SD over SPI driver configuration structure.

```
#include <mmc_spi.h>
```

Collaboration diagram for MMCConfig:



Data Fields

- `SPIDriver * spip`

SPI driver associated to this MMC driver.

- const **SPIConfig** * lscfg
SPI low speed configuration used during initialization.
- const **SPIConfig** * hscfg
SPI high speed configuration used during transfers.

12.67.2 Field Documentation

12.67.2.1 **SPIDriver*** MMCConfig::spip

SPI driver associated to this MMC driver.

12.67.2.2 const **SPIConfig*** MMCConfig::lscfg

SPI low speed configuration used during initialization.

12.67.2.3 const **SPIConfig*** MMCConfig::hscfg

SPI high speed configuration used during transfers.

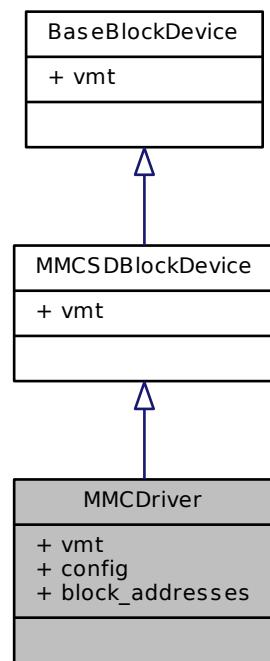
12.68 MMCDriver Struct Reference

12.68.1 Detailed Description

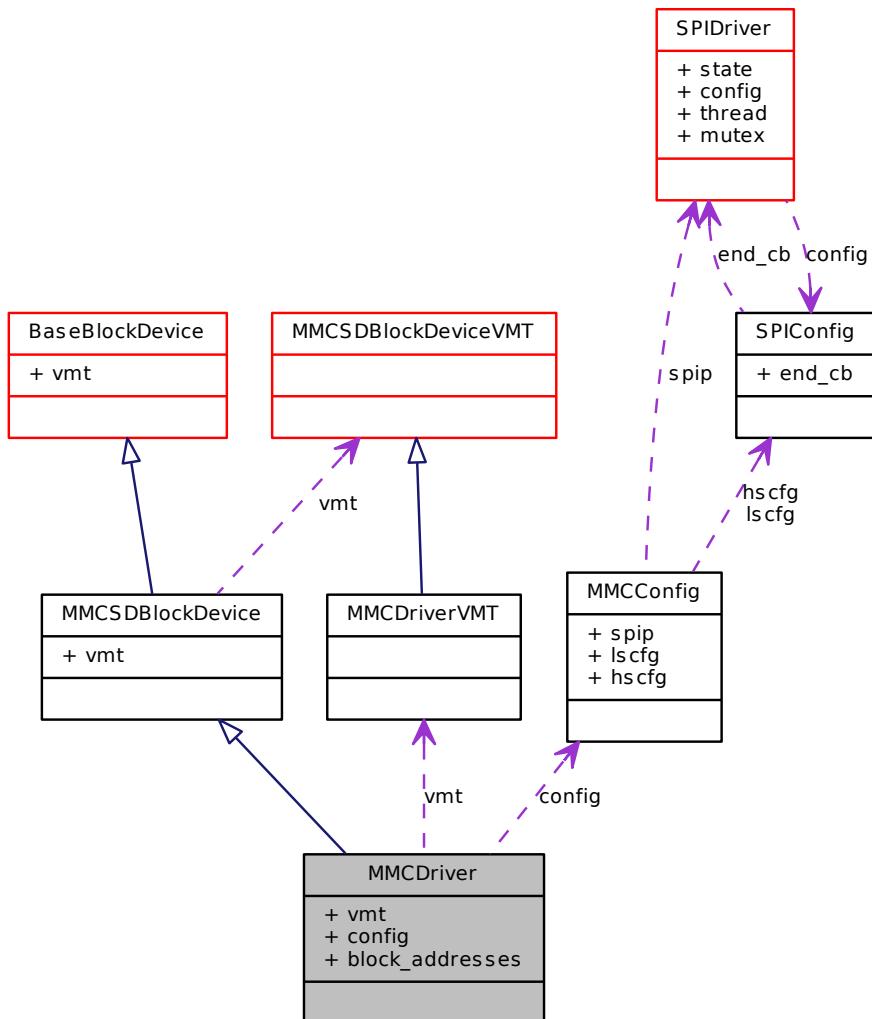
Structure representing a MMC/SD over SPI driver.

```
#include <mmc_spi.h>
```

Inheritance diagram for MMCDriver:



Collaboration diagram for MMCDriver:



Data Fields

- struct [MMCDriverVMT](#) * **vmt**
Virtual Methods Table.
- [_mmcsd_block_device_data](#) const [MMCCConfig](#) * **config**
Current configuration data.

12.68.2 Field Documentation

12.68.2.1 struct MMCDriverVMT* MMCDriver::vmt

Virtual Methods Table.

Reimplemented from [MMCSDBlockDevice](#).

12.68.2.2 _mmcsd_block_device_data const MMCConfig* MMCDriver::config

Current configuration data.

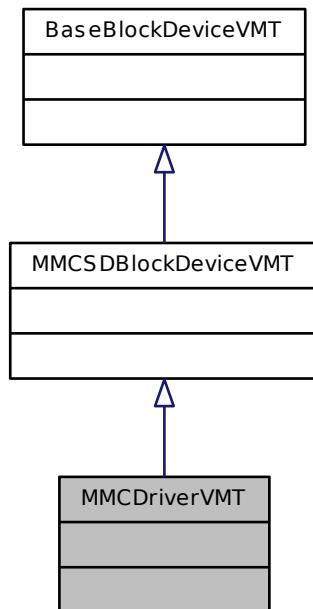
12.69 MMCDriverVMT Struct Reference

12.69.1 Detailed Description

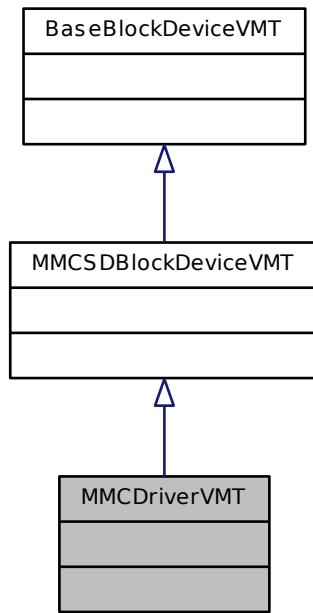
[MMCDriver](#) virtual methods table.

```
#include <mmc_spi.h>
```

Inheritance diagram for MMCDriverVMT:



Collaboration diagram for MMCBlockDeviceVMT:



12.70 MMCSDBlockDevice Struct Reference

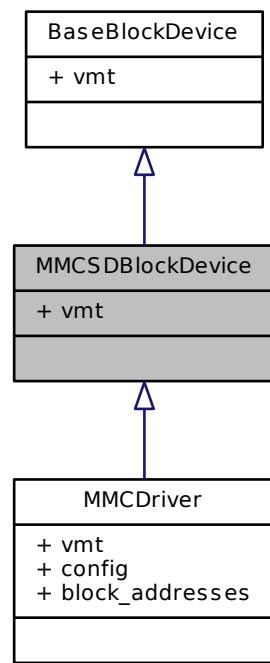
12.70.1 Detailed Description

MMC/SD block device class.

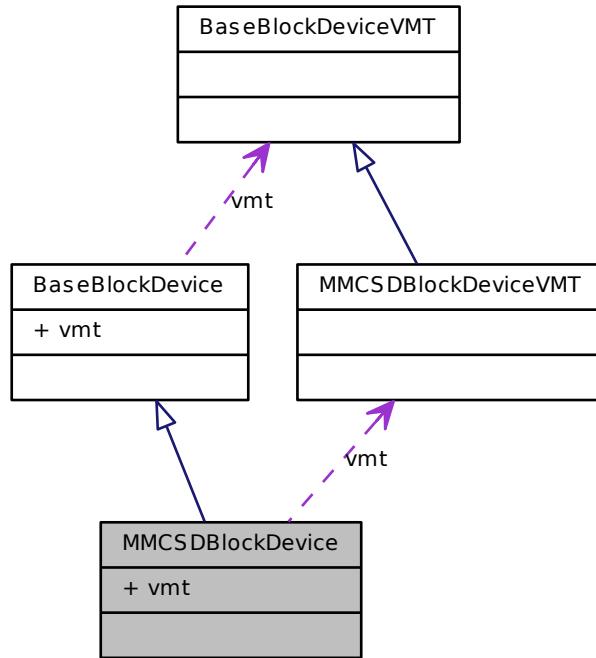
This class represents a, block-accessible, MMC/SD device.

```
#include <mmcsd.h>
```

Inheritance diagram for MMCSDBlockDevice:



Collaboration diagram for MMCSDBlockDevice:



Data Fields

- struct [MMCSDBlockDeviceVMT](#) * `vmt`

Virtual Methods Table.

12.70.2 Field Documentation

12.70.2.1 struct MMCSDBlockDeviceVMT* MMCSDBlockDevice::vmt

Virtual Methods Table.

Reimplemented from [BaseBlockDevice](#).

Reimplemented in [MMCDriver](#).

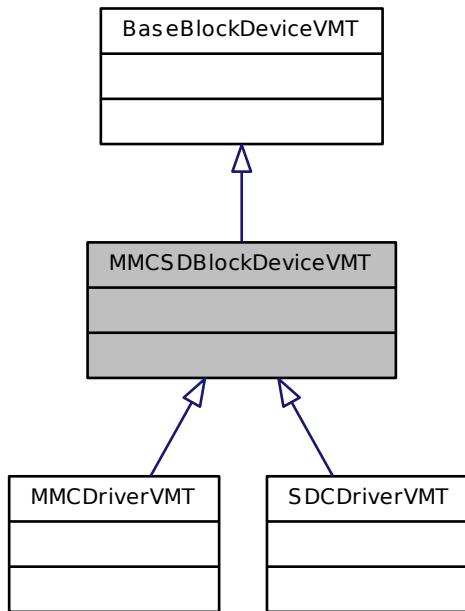
12.71 MMCSDBlockDeviceVMT Struct Reference

12.71.1 Detailed Description

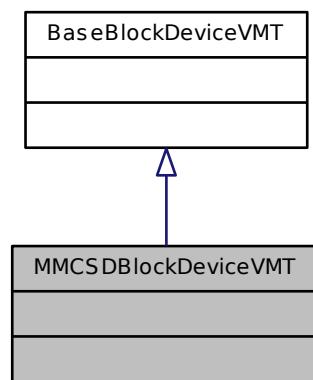
[MMCSDBlockDevice](#) virtual methods table.

```
#include <mmcsd.h>
```

Inheritance diagram for MMCSDBlockDeviceVMT:



Collaboration diagram for MMCSDBlockDeviceVMT:



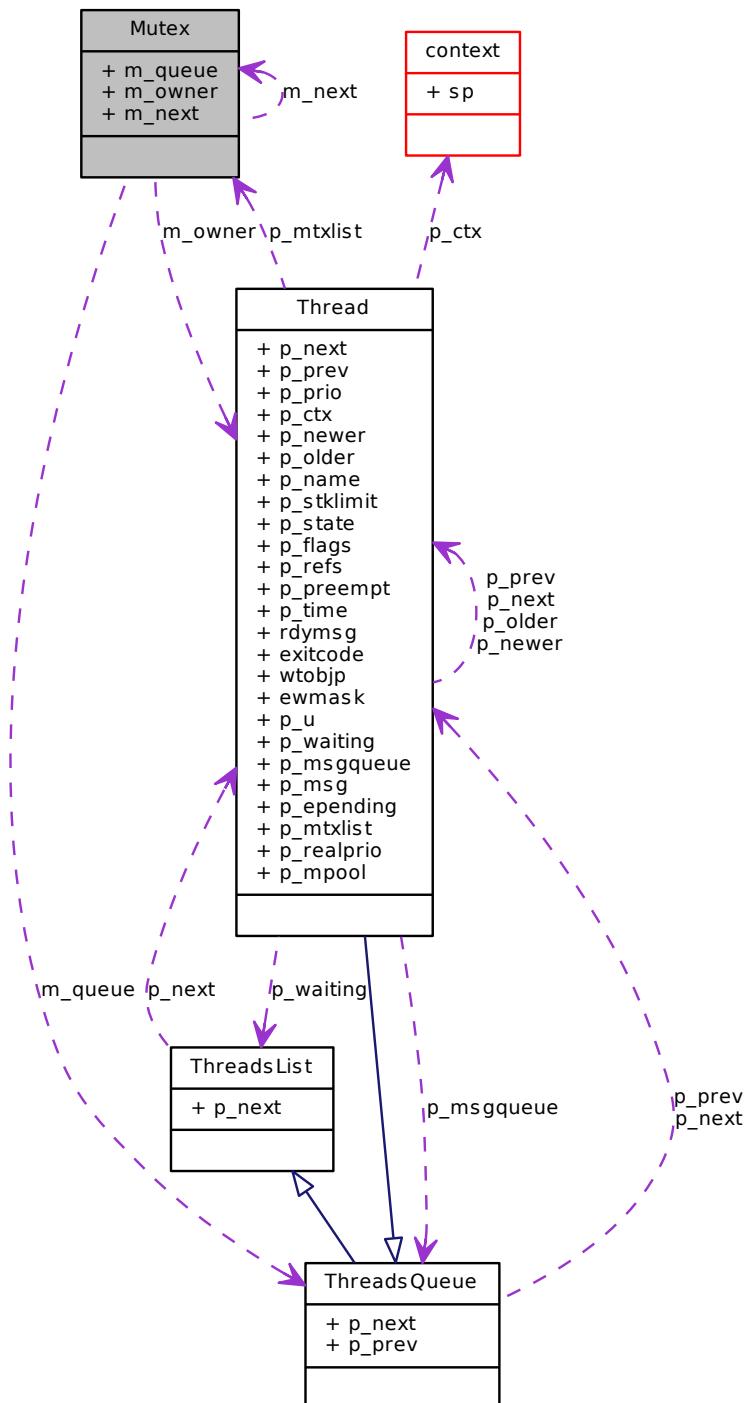
12.72 Mutex Struct Reference

12.72.1 Detailed Description

[Mutex](#) structure.

```
#include <chmtx.h>
```

Collaboration diagram for Mutex:



Data Fields

- **ThreadsQueue m_queue**
Queue of the threads sleeping on this Mutex.
- **Thread * m_owner**
Owner Thread pointer or NULL.
- struct **Mutex * m_next**
Next Mutex into an owner-list or NULL.

12.72.2 Field Documentation

12.72.2.1 ThreadsQueue Mutex::m_queue

Queue of the threads sleeping on this Mutex.

12.72.2.2 Thread* Mutex::m_owner

Owner Thread pointer or NULL.

12.72.2.3 struct Mutex* Mutex::m_next

Next Mutex into an owner-list or NULL.

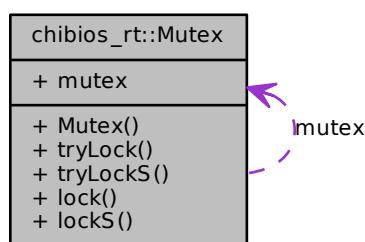
12.73 chibios_rt::Mutex Class Reference

12.73.1 Detailed Description

Class encapsulating a mutex.

```
#include <ch.hpp>
```

Collaboration diagram for chibios_rt::Mutex:



Public Member Functions

- **Mutex (void)**
Mutex object constructor.

- bool [tryLock](#) (void)
Tries to lock a mutex.
- bool [tryLockS](#) (void)
Tries to lock a mutex.
- void [lock](#) (void)
Locks the specified mutex.
- void [lockS](#) (void)
Locks the specified mutex.

Data Fields

- [::Mutex mutex](#)
Embedded [Mutex](#) structure.

12.73.2 Constructor & Destructor Documentation

12.73.2.1 Mutex::Mutex(void)

[Mutex](#) object constructor.

The embedded [Mutex](#) structure is initialized.

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:



12.73.3 Member Function Documentation

12.73.3.1 bool Mutex::tryLock(void)

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.

Returns

The operation status.

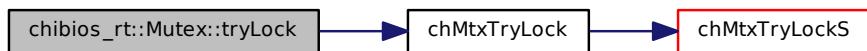
Return values

<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:

**12.73.3.2 bool Mutex::tryLockS (void)**

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.

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:



12.73.3.3 void Mutex::lock (void)

Locks the specified mutex.

Postcondition

The mutex is locked and inserted in the per-thread stack of owned mutexes.

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:



12.73.3.4 void Mutex::lockS (void)

Locks the specified mutex.

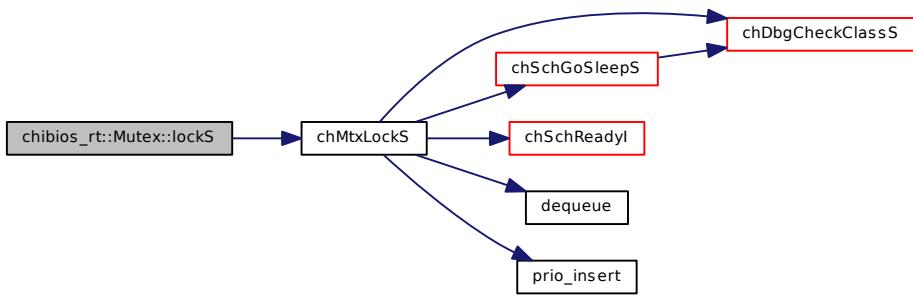
Postcondition

The mutex is locked and inserted in the per-thread stack of owned mutexes.

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:



12.73.4 Field Documentation

12.73.4.1 ::Mutex chibios_rt::Mutex::mutex

Embedded [Mutex](#) structure.

12.74 chibios_rt::ObjectsPool< T, N > Class Template Reference

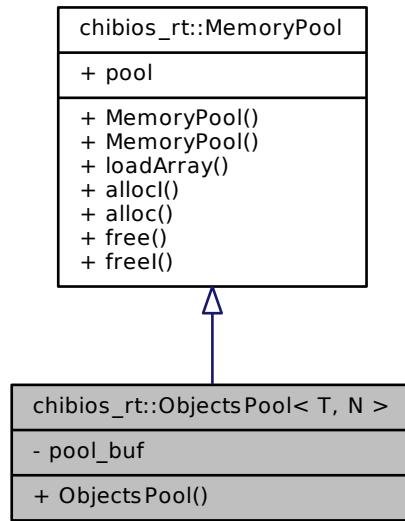
12.74.1 Detailed Description

```
template<class T, size_t N>class chibios_rt::ObjectsPool< T, N >
```

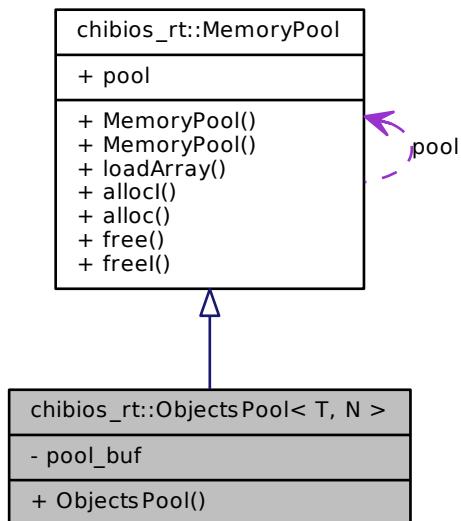
Template class encapsulating a memory pool and its elements.

```
#include <ch.hpp>
```

Inheritance diagram for chibios_rt::ObjectsPool< T, N >:



Collaboration diagram for chibios_rt::ObjectsPool< T, N >:



Public Member Functions

- [ObjectsPool](#) (void)
ObjectsPool constructor.

12.74.2 Constructor & Destructor Documentation

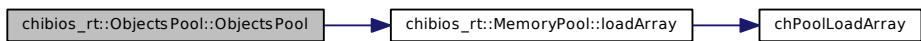
12.74.2.1 template<class T , size_t N> chibios_rt::ObjectsPool< T,N >::ObjectsPool(void) [inline]

[ObjectsPool](#) constructor.

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:



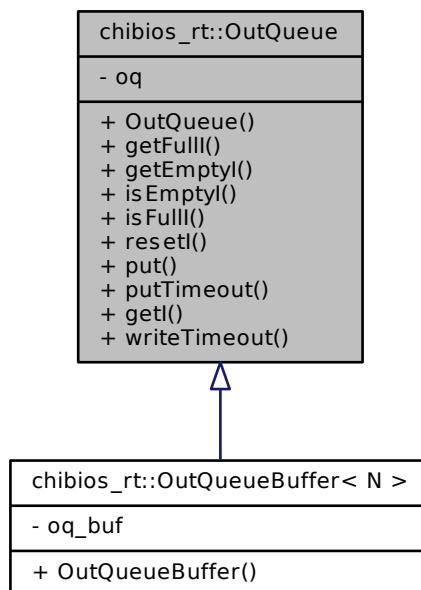
12.75 chibios_rt::OutQueue Class Reference

12.75.1 Detailed Description

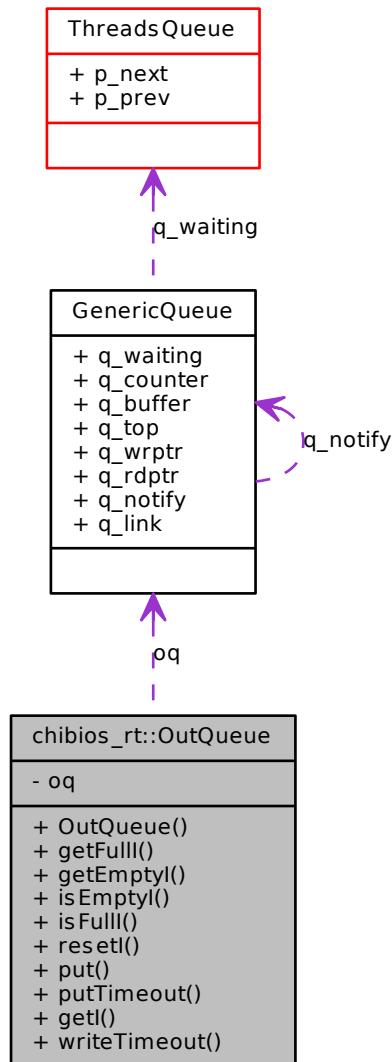
Class encapsulating an output queue.

```
#include <ch.hpp>
```

Inheritance diagram for chibios_rt::OutQueue:



Collaboration diagram for chibios_rt::OutQueue:



Public Member Functions

- `OutQueue (uint8_t *bp, size_t size, qnotify_t onfy, void *link)`
OutQueue constructor.
- `size_t getFulll ()`
Returns the filled space into an output queue.
- `size_t getEmptyl ()`
Returns the empty space into an output queue.
- `bool isEmptyl ()`
Evaluates to TRUE if the specified output queue is empty.
- `bool isFulll ()`
Evaluates to TRUE if the specified output queue is full.

- void [resetl](#) (void)
Resets an output queue.
- [msg_t put](#) (uint8_t b)
Output queue write.
- [msg_t putTimeout](#) (uint8_t b, [systime_t](#) time)
Output queue write with timeout.
- [msg_t getl](#) (void)
Output queue read.
- [size_t writeTimeout](#) (const uint8_t *bp, size_t n, [systime_t](#) time)
Output queue write with timeout.

12.75.2 Constructor & Destructor Documentation

12.75.2.1 chibios_rt::OutQueue::OutQueue ([uint8_t * bp](#), [size_t size](#), [qnotify_t onfy](#), [void * link](#))

[OutQueue](#) constructor.

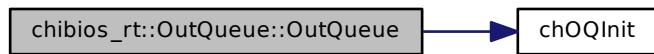
Parameters

in	<i>bp</i>	pointer to a memory area allocated as queue buffer
in	<i>size</i>	size of the queue buffer
in	<i>onfy</i>	pointer to a callback function that is invoked when data is written to the queue. The value can be <code>NULL</code> .
in	<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:



12.75.3 Member Function Documentation

12.75.3.1 [size_t chibios_rt::OutQueue::getFulll](#) ([void](#))

Returns the filled space into an output queue.

Returns

The number of full bytes in the queue.

Return values

0 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.

12.75.3.2 size_t chibios_rt::OutQueue::getEmpty() (void)

Returns the empty space into an output queue.

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.

12.75.3.3 bool chibios_rt::OutQueue::isEmpty() (void)

Evaluates to TRUE if the specified output queue is empty.

Returns

The queue status.

Return values

false if the queue is not empty.
true 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.

12.75.3.4 bool chibios_rt::OutQueue::isFull() (void)

Evaluates to TRUE if the specified output queue is full.

Returns

The queue status.

Return values

FALSE if the queue is not full.
TRUE 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.

12.75.3.5 void chibios_rt::OutQueue::resetl (void)

Resets an output queue.

All the data in the output queue is erased and lost, any waiting thread is resumed with status Q_RESET.

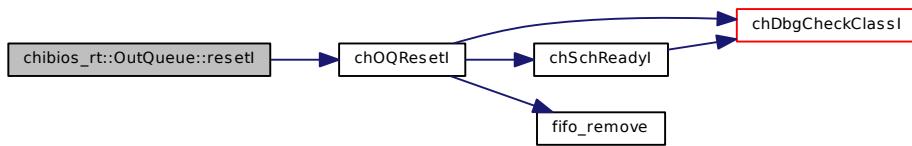
Note

A reset operation can be used by a low level driver in order to obtain immediate attention from the high level layers.

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:



12.75.3.6 msg_t chibios_rt::OutQueue::put (uint8_t b)

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>b</i> the byte value to be written in the queue
----	--

Returns

The operation status.

Return values

<i>Q_OK</i>	if the operation succeeded.
<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.

12.75.3.7 msg_t chibios_rt::OutQueue::putTimeout (uint8_t b, systime_t time)

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>b</i> the byte value to be written in the queue
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

The operation status.

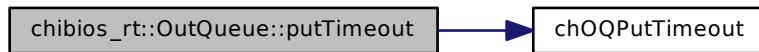
Return values

<i>Q_OK</i>	if the operation succeeded.
<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:

**12.75.3.8 msg_t chibios_rt::OutQueue::getl(void)**

Output queue read.

A byte value is read from the low end of an output queue.

Returns

The byte value from the queue.

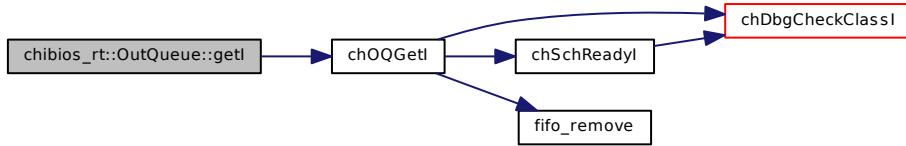
Return values

<i>Q_EMPTY</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:



12.75.3.9 size_t chibios_rt::OutQueue::writeTimeout (const uint8_t * bp, size_t n, systime_t time)

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

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>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.

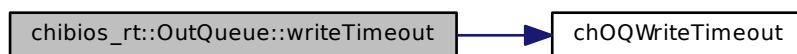
Returns

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:



12.76 chibios_rt::OutQueueBuffer< N > Class Template Reference

12.76.1 Detailed Description

```
template<int N>class chibios_rt::OutQueueBuffer< N >
```

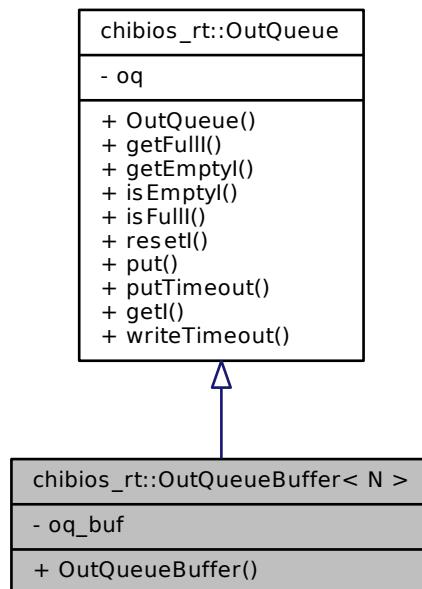
Template class encapsulating an output queue and its buffer.

Parameters

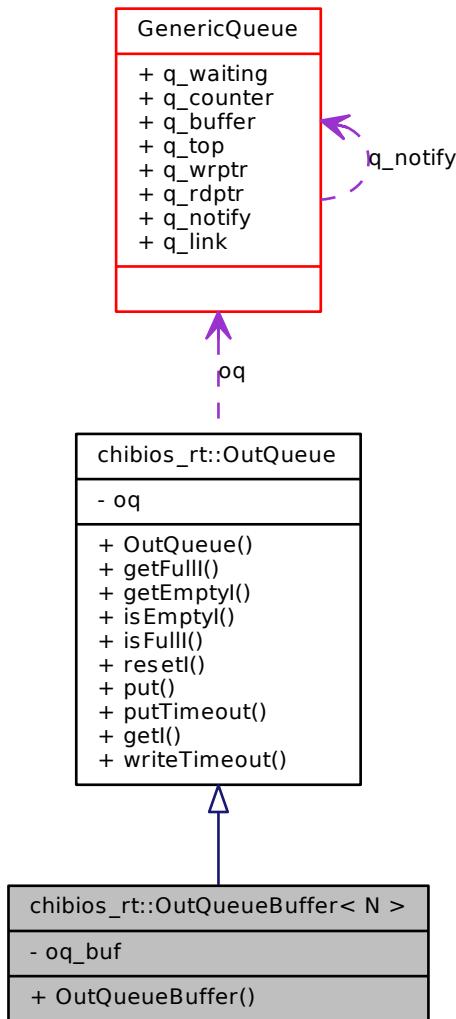
N size of the output queue

```
#include <ch.hpp>
```

Inheritance diagram for chibios_rt::OutQueueBuffer< N >:



Collaboration diagram for chibios_rt::OutQueueBuffer< N >:



Public Member Functions

- [OutQueueBuffer \(qnotify_t onfy, void *link\)](#)
OutQueueBuffer constructor.

12.76.2 Constructor & Destructor Documentation

12.76.2.1 template<int N> chibios_rt::OutQueueBuffer< N >::OutQueueBuffer (qnotify_t onfy, void * link) [inline]

[OutQueueBuffer constructor.](#)

Parameters

in	onfy	output notify callback function
in	link	parameter to be passed to the callback

Function Class:

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

12.77 PALConfig Struct Reference

12.77.1 Detailed Description

Generic I/O ports static initializer.

An instance of this structure must be passed to [palInit\(\)](#) at system startup time in order to initialize the digital I/O subsystem. This represents only the initial setup, specific pads or whole ports can be reprogrammed at later time.

Note

Implementations may extend this structure to contain more, architecture dependent, fields.

```
#include <pal_ll1d.h>
```

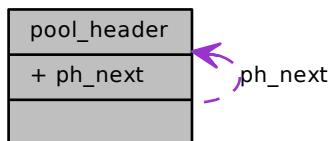
12.78 pool_header Struct Reference

12.78.1 Detailed Description

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.

12.78.2 Field Documentation

12.78.2.1 struct pool_header* pool_header::ph_next

Pointer to the next pool header in the list.

12.79 PWMChannelConfig Struct Reference

12.79.1 Detailed Description

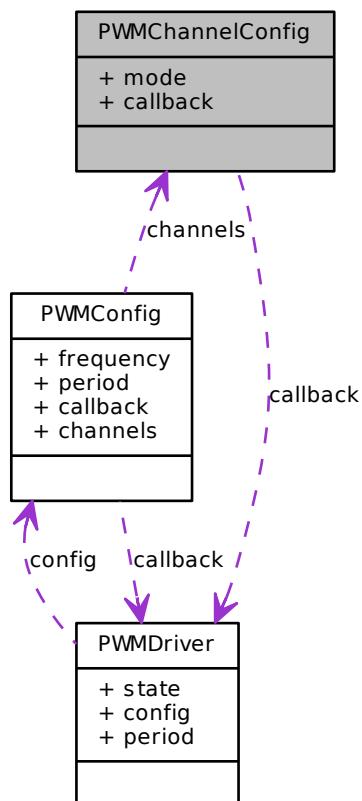
PWM driver channel configuration structure.

Note

Some architectures may not be able to support the channel mode or the callback, in this case the fields are ignored.

```
#include <pwm_ll.h>
```

Collaboration diagram for PWMChannelConfig:



Data Fields

- [pwmmode_t mode](#)
Channel active logic level.
- [pwmcallback_t callback](#)
Channel callback pointer.

12.79.2 Field Documentation

12.79.2.1 pwmmode_t PWMChannelConfig::mode

Channel active logic level.

12.79.2.2 pwmcallback_t PWMChannelConfig::callback

Channel callback pointer.

Note

This callback is invoked on the channel compare event. If set to `NULL` then the callback is disabled.

12.80 PWMConfig Struct Reference

12.80.1 Detailed Description

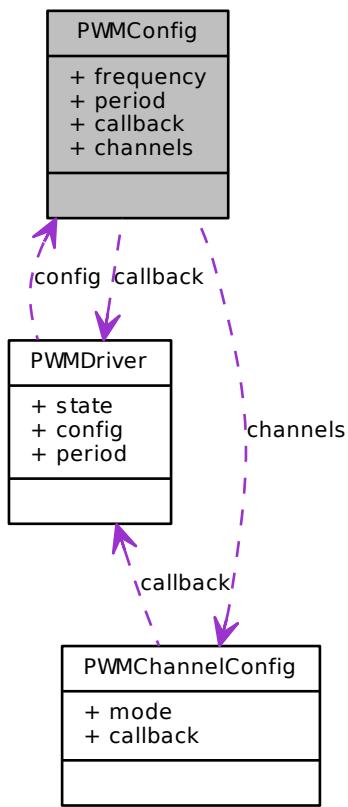
Driver configuration structure.

Note

Implementations may extend this structure to contain more, architecture dependent, fields.

```
#include <pwm_llld.h>
```

Collaboration diagram for PWMConfig:



Data Fields

- `uint32_t frequency`
Timer clock in Hz.
- `pwmcnt_t period`
PWM period in ticks.
- `pwmcallback_t callback`
Periodic callback pointer.
- `PWMChannelConfig channels [PWM_CHANNELS]`
Channels configurations.

12.80.2 Field Documentation

12.80.2.1 `uint32_t PWMConfig::frequency`

Timer clock in Hz.

Note

The low level can use assertions in order to catch invalid frequency specifications.

12.80.2.2 pwmcnt_t PWMConfig::period

PWM period in ticks.

Note

The low level can use assertions in order to catch invalid period specifications.

12.80.2.3 pwmcallback_t PWMConfig::callback

Periodic callback pointer.

Note

This callback is invoked on PWM counter reset. If set to `NULL` then the callback is disabled.

12.80.2.4 PWMChannelConfig PWMConfig::channels[PWM_CHANNELS]

Channels configurations.

12.81 PWMDriver Struct Reference

12.81.1 Detailed Description

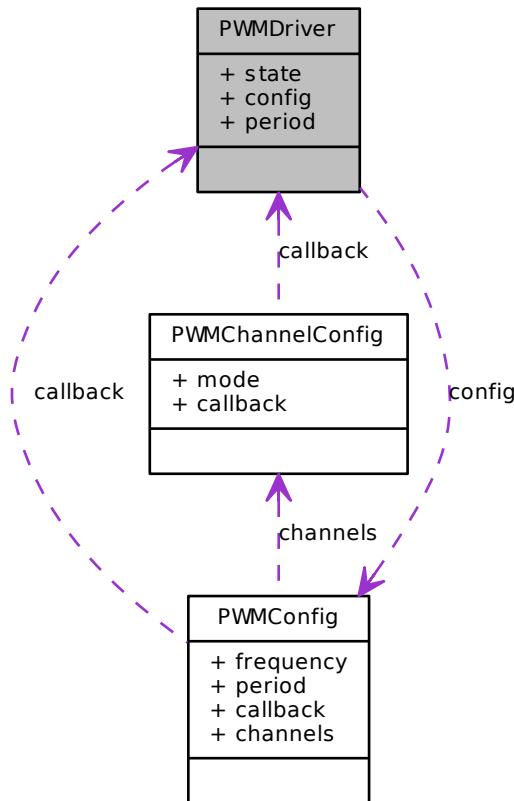
Structure representing an PWM driver.

Note

Implementations may extend this structure to contain more, architecture dependent, fields.

```
#include <pwm_lld.h>
```

Collaboration diagram for PWMDriver:



Data Fields

- `pwmstate_t state`
Driver state.
- `const PWMConfig * config`
Current configuration data.
- `pwmcnt_t period`
Current PWM period in ticks.

12.81.2 Field Documentation

12.81.2.1 `pwmstate_t PWMDriver::state`

Driver state.

12.81.2.2 `const PWMConfig* PWMDriver::config`

Current configuration data.

12.81.2.3 pwmcnt_t PWMDriver::period

Current PWM period in ticks.

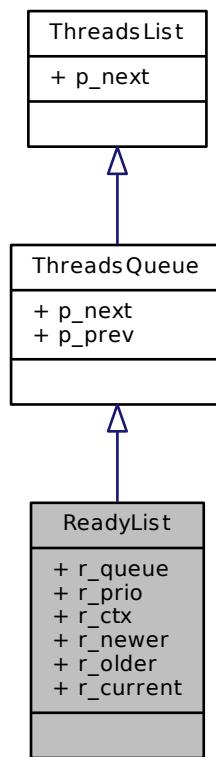
12.82 ReadyList Struct Reference

12.82.1 Detailed Description

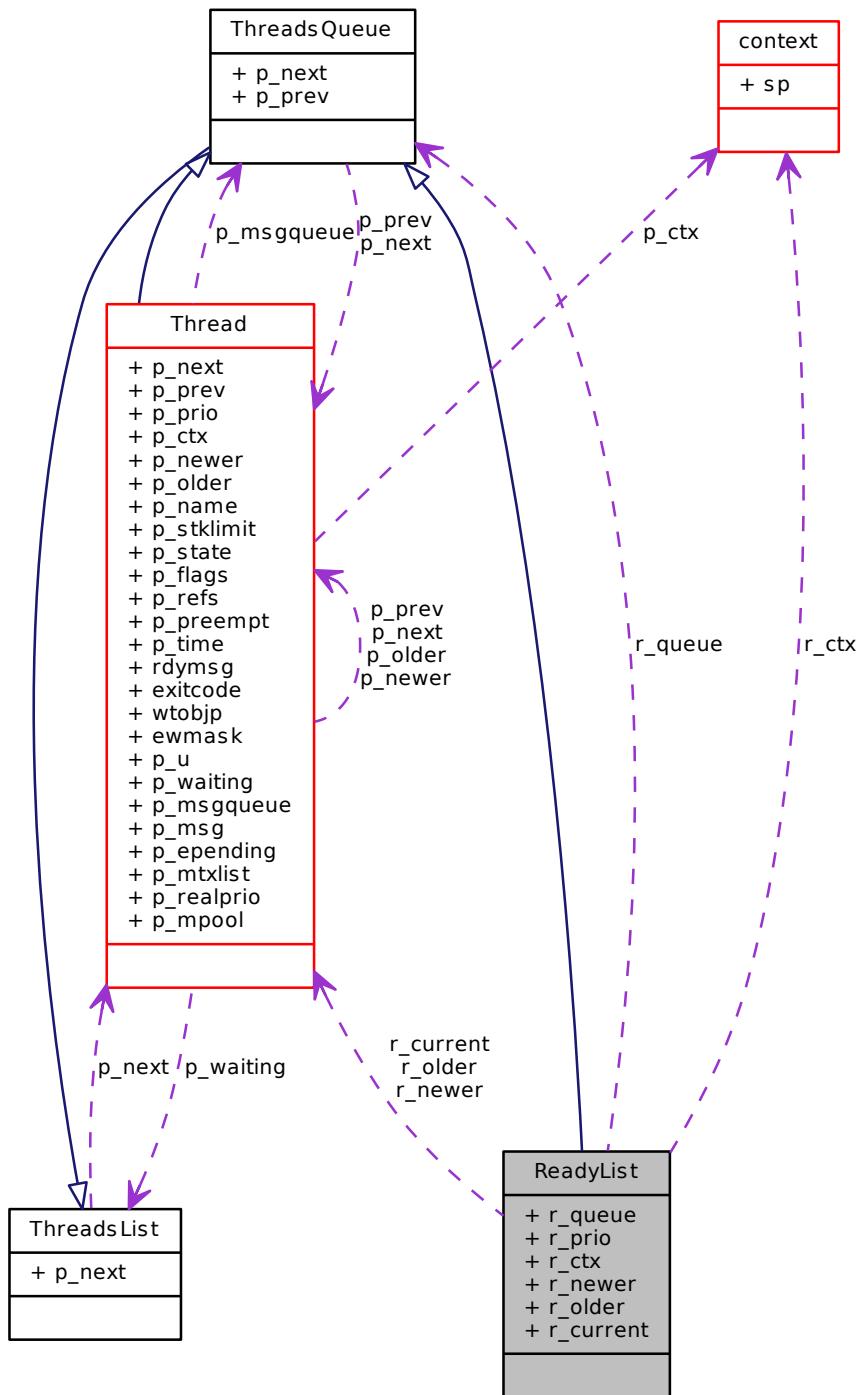
Ready list header.

```
#include <chsched.h>
```

Inheritance diagram for ReadyList:



Collaboration diagram for ReadyList:



Data Fields

- `ThreadsQueue r_queue`

Threads queue.

- `tprio_t r_prio`
This field must be initialized to zero.
- struct `context r_ctx`
Not used, present because offsets.
- `Thread * r_newer`
Newer registry element.
- `Thread * r_older`
Older registry element.
- `Thread * r_current`
The currently running thread.

12.82.2 Field Documentation

12.82.2.1 ThreadsQueue ReadyList::r_queue

Threads queue.

12.82.2.2 tprio_t ReadyList::r_prio

This field must be initialized to zero.

12.82.2.3 struct context ReadyList::r_ctx

Not used, present because offsets.

12.82.2.4 Thread* ReadyList::r_newer

Newer registry element.

12.82.2.5 Thread* ReadyList::r_older

Older registry element.

12.82.2.6 Thread* ReadyList::r_current

The currently running thread.

12.83 SDCCConfig Struct Reference

12.83.1 Detailed Description

Driver configuration structure.

Note

It could be empty on some architectures.

```
#include <sdc_lld.h>
```

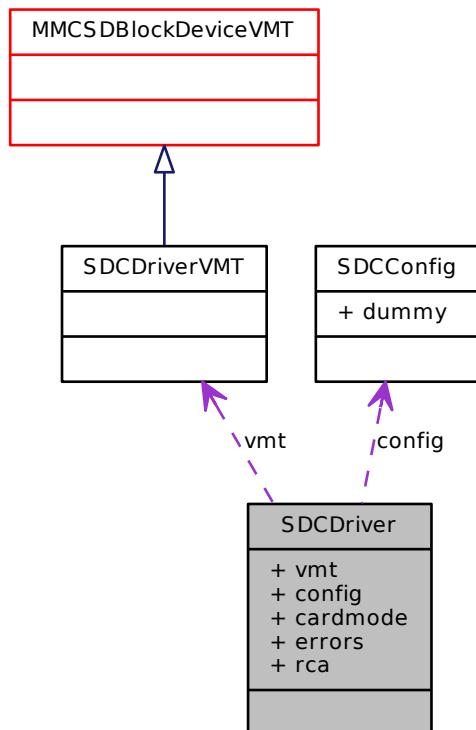
12.84 SDCDriver Struct Reference

12.84.1 Detailed Description

Structure representing an SDC driver.

```
#include <sdc_ll.h>
```

Collaboration diagram for SDCDriver:



Data Fields

- struct `SDCDriverVMT` * `vmt`
Virtual Methods Table.
- `_mmcsd_block_device_data` const `SDCConfig` * `config`
Current configuration data.
- `sdcmode_t` `cardmode`
Various flags regarding the mounted card.
- `sdcflags_t` `errors`
Errors flags.
- `uint32_t` `rca`
Card RCA.

12.84.2 Field Documentation

12.84.2.1 `struct SDCDriverVMT* SDCDriver::vmt`

Virtual Methods Table.

12.84.2.2 `_mmcblk_device_data const SDCCConfig* SDCDriver::config`

Current configuration data.

12.84.2.3 `sdcmode_t SDCDriver::cardmode`

Various flags regarding the mounted card.

12.84.2.4 `sdcflags_t SDCDriver::errors`

Errors flags.

12.84.2.5 `uint32_t SDCDriver::rca`

Card RCA.

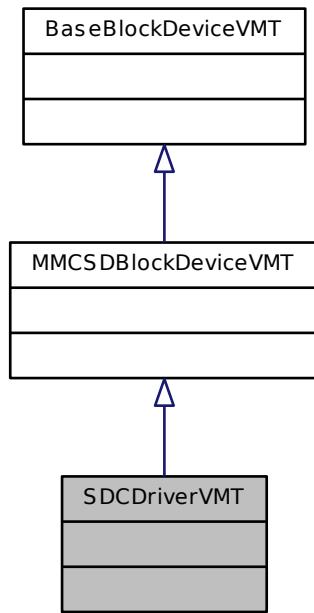
12.85 SDCDriverVMT Struct Reference

12.85.1 Detailed Description

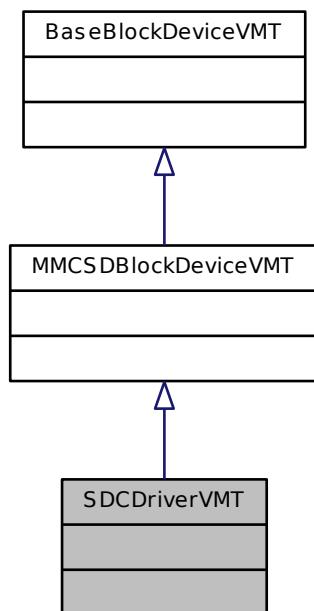
`SDCDriver` virtual methods table.

```
#include <sdc_ll.h>
```

Inheritance diagram for SDCDriverVMT:



Collaboration diagram for SDCDriverVMT:



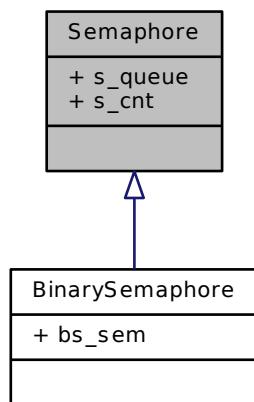
12.86 Semaphore Struct Reference

12.86.1 Detailed Description

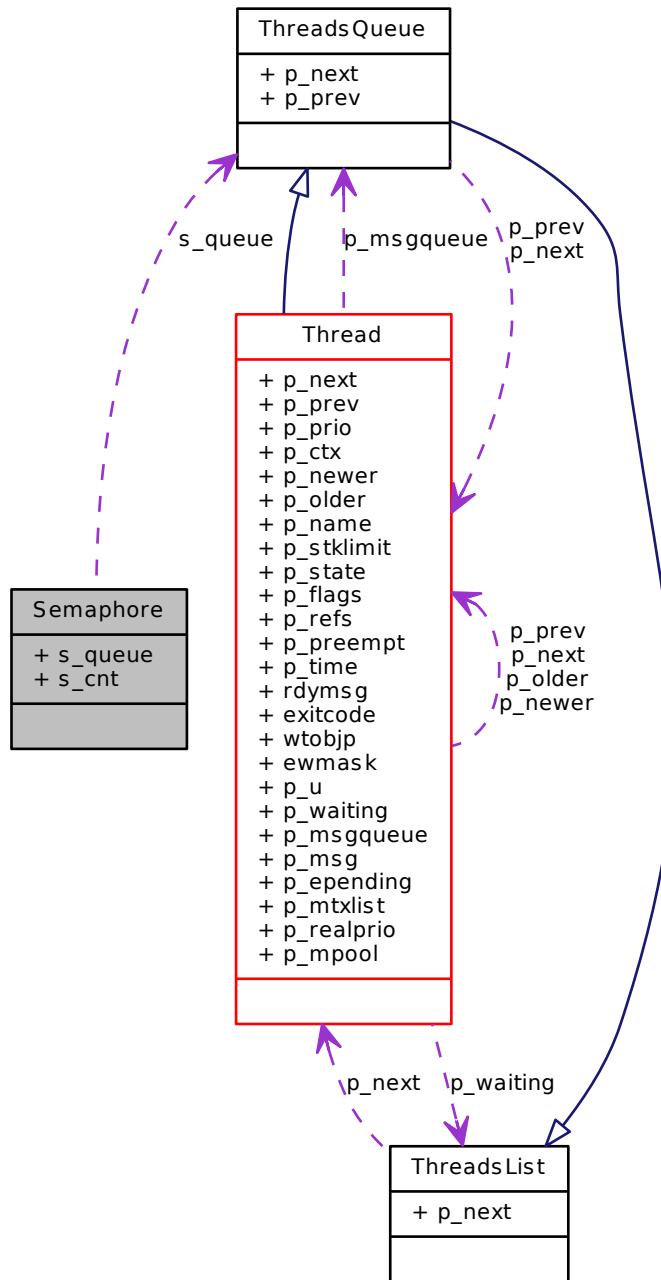
[Semaphore](#) structure.

```
#include <chsem.h>
```

Inheritance diagram for Semaphore:



Collaboration diagram for Semaphore:



Data Fields

- **ThreadsQueue s_queue**
Queue of the threads sleeping on this semaphore.
- **cnt_t s_cnt**
The semaphore counter.

12.86.2 Field Documentation

12.86.2.1 ThreadsQueue Semaphore::s_queue

Queue of the threads sleeping on this semaphore.

12.86.2.2 cnt_t Semaphore::s_cnt

The semaphore counter.

12.87 SerialConfig Struct Reference

12.87.1 Detailed Description

Generic Serial Driver configuration structure.

An instance of this structure must be passed to [sdStart \(\)](#) in order to configure and start a serial driver operations.

Note

Implementations may extend this structure to contain more, architecture dependent, fields.

```
#include <serial_lld.h>
```

Data Fields

- `uint32_t sc_speed`

Bit rate.

12.87.2 Field Documentation

12.87.2.1 uint32_t SerialConfig::sc_speed

Bit rate.

12.88 SerialDriver Struct Reference

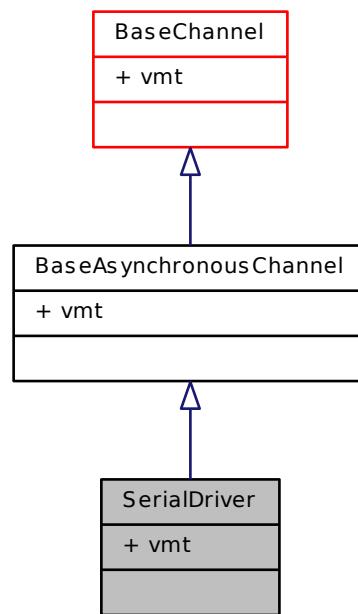
12.88.1 Detailed Description

Full duplex serial driver class.

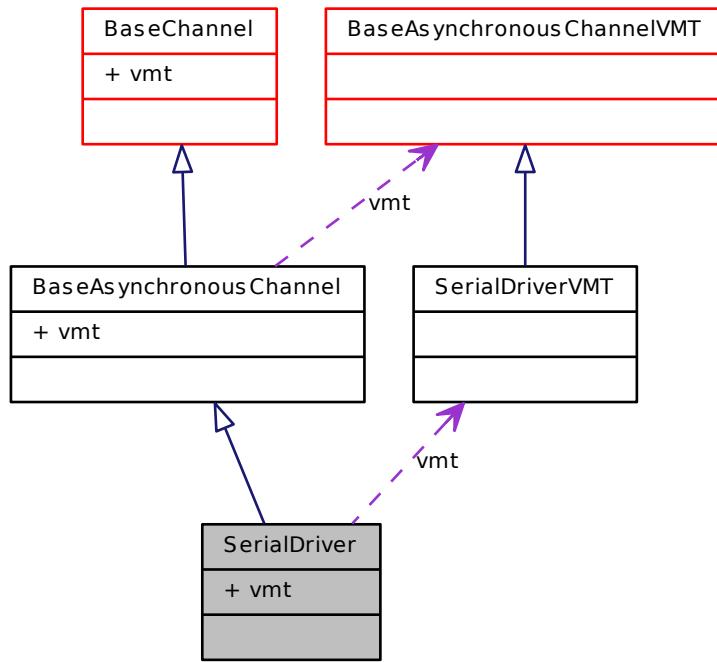
This class extends [BaseAsynchronousChannel](#) by adding physical I/O queues.

```
#include <serial.h>
```

Inheritance diagram for SerialDriver:



Collaboration diagram for SerialDriver:



Data Fields

- struct [SerialDriverVMT](#) * vmt

Virtual Methods Table.

12.88.2 Field Documentation

12.88.2.1 struct [SerialDriverVMT](#)* SerialDriver::vmt

Virtual Methods Table.

Reimplemented from [BaseAsynchronousChannel](#).

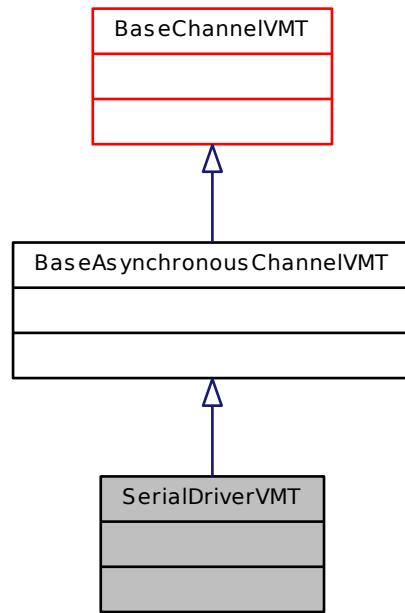
12.89 SerialDriverVMT Struct Reference

12.89.1 Detailed Description

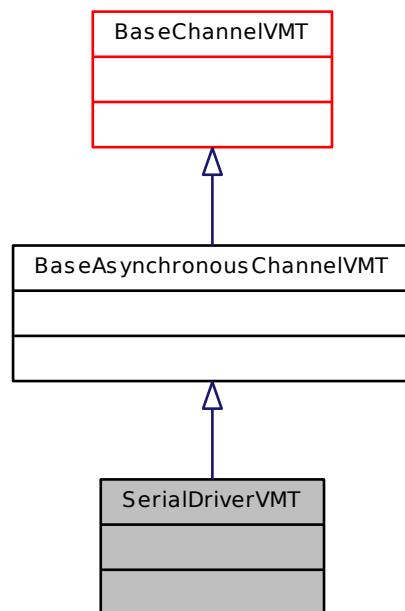
[SerialDriver](#) virtual methods table.

```
#include <serial.h>
```

Inheritance diagram for SerialDriverVMT:



Collaboration diagram for SerialDriverVMT:



12.90 SerialUSBConfig Struct Reference

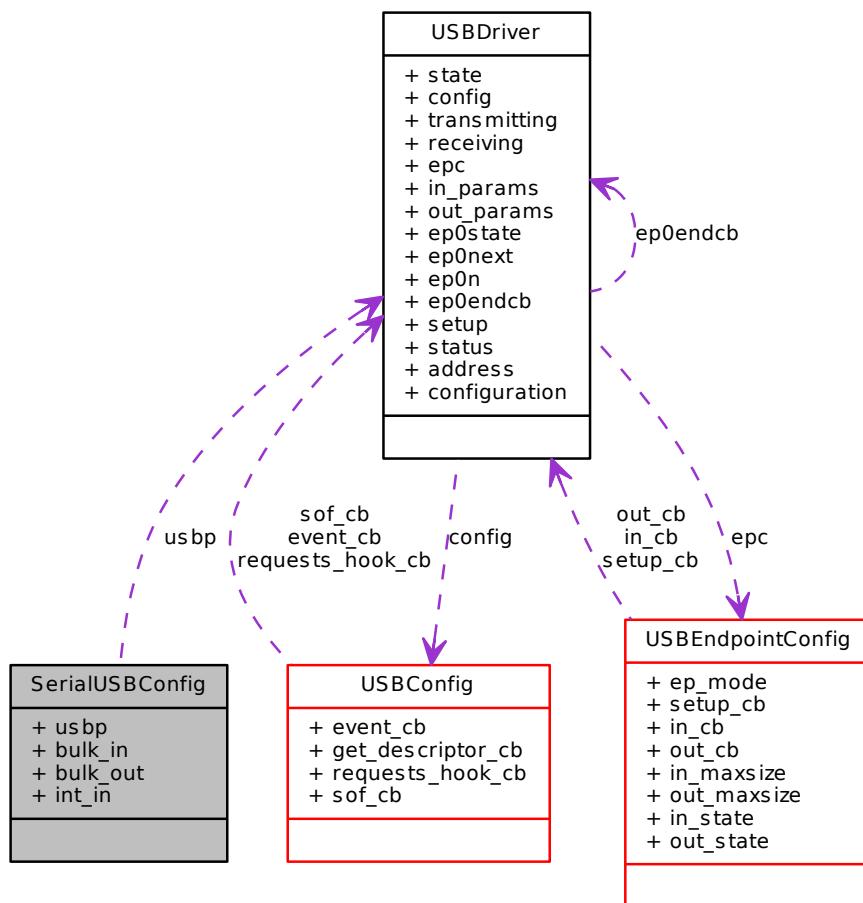
12.90.1 Detailed Description

Serial over USB Driver configuration structure.

An instance of this structure must be passed to `sduStart()` in order to configure and start the driver operations.

```
#include <serial_usb.h>
```

Collaboration diagram for SerialUSBConfig:



Data Fields

- **USBDriver * usbp**
USB driver to use.
- **usbep_t bulk_in**
Bulk IN endpoint used for outgoing data transfer.
- **usbep_t bulk_out**
Bulk OUT endpoint used for incoming data transfer.
- **usbep_t int_in**
Interrupt IN endpoint used for notifications.

12.90.2 Field Documentation

12.90.2.1 `USBDriver* SerialUSBConfig::usbp`

USB driver to use.

12.90.2.2 `usbep_t SerialUSBConfig::bulk_in`

Bulk IN endpoint used for outgoing data transfer.

12.90.2.3 `usbep_t SerialUSBConfig::bulk_out`

Bulk OUT endpoint used for incoming data transfer.

12.90.2.4 `usbep_t SerialUSBConfig::int_in`

Interrupt IN endpoint used for notifications.

12.91 SerialUSBDriver Struct Reference

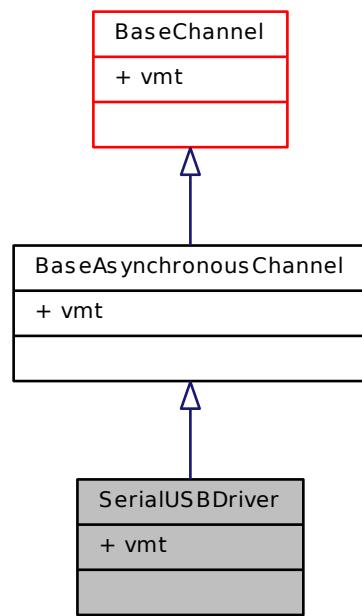
12.91.1 Detailed Description

Full duplex serial driver class.

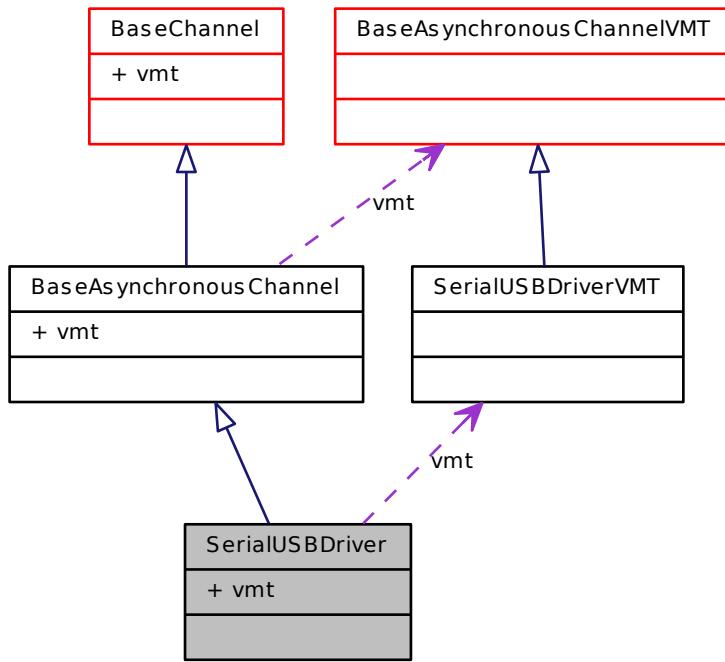
This class extends [BaseAsynchronousChannel](#) by adding physical I/O queues.

```
#include <serial_usb.h>
```

Inheritance diagram for SerialUSBDriver:



Collaboration diagram for SerialUSBDriver:



Data Fields

- struct [SerialUSBDriverVMT](#) * **vmt**

Virtual Methods Table.

12.91.2 Field Documentation

12.91.2.1 struct [SerialUSBDriverVMT](#)* **SerialUSBDriver::vmt**

Virtual Methods Table.

Reimplemented from [BaseAsynchronousChannel](#).

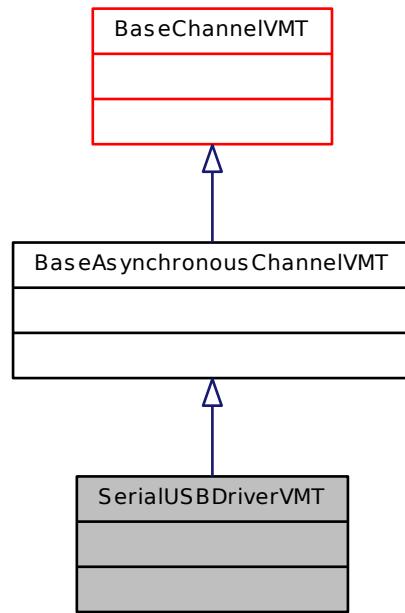
12.92 SerialUSBDriverVMT Struct Reference

12.92.1 Detailed Description

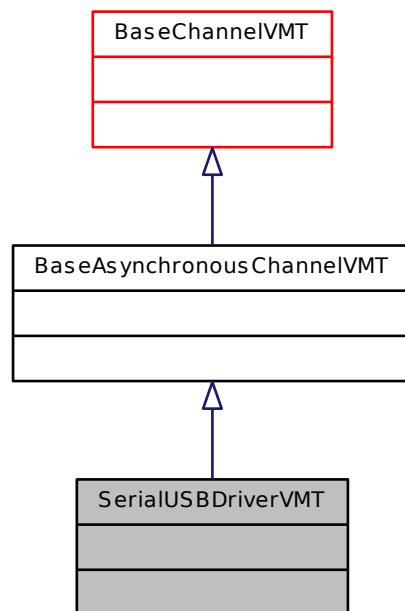
[SerialDriver](#) virtual methods table.

```
#include <serial_usb.h>
```

Inheritance diagram for SerialUSBDriverVMT:



Collaboration diagram for SerialUSBDriverVMT:



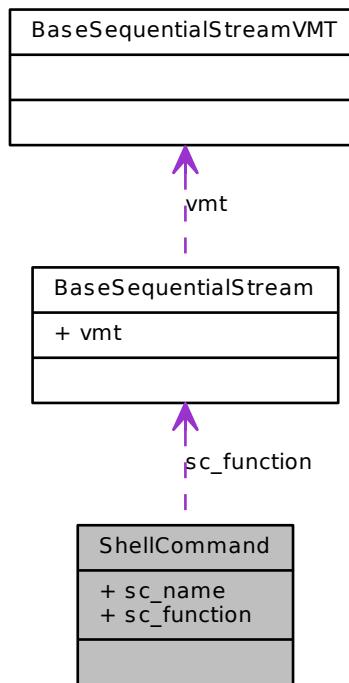
12.93 ShellCommand Struct Reference

12.93.1 Detailed Description

Custom command entry type.

```
#include <shell.h>
```

Collaboration diagram for ShellCommand:



Data Fields

- `const char * sc_name`
Command name.
- `shellcmd_t sc_function`
Command function.

12.93.2 Field Documentation

12.93.2.1 `const char* ShellCommand::sc_name`

Command name.

12.93.2.2 `shellcmd_t ShellCommand::sc_function`

Command function.

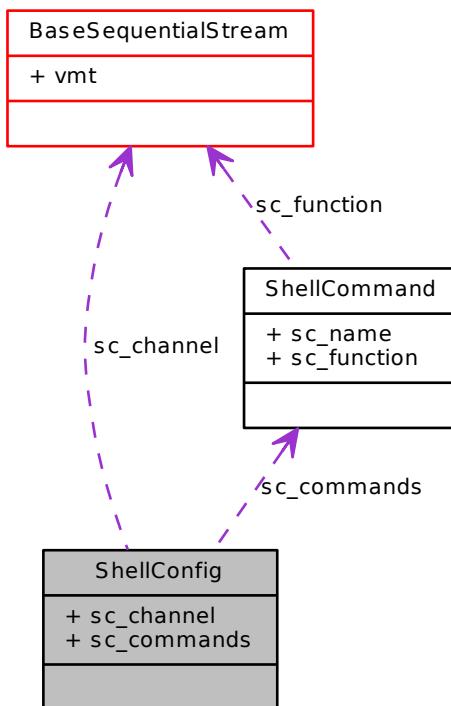
12.94 ShellConfig Struct Reference

12.94.1 Detailed Description

Shell descriptor type.

```
#include <shell.h>
```

Collaboration diagram for ShellConfig:



Data Fields

- `BaseSequentialStream * sc_channel`
I/O channel associated to the shell.
- `const ShellCommand * sc_commands`
Shell extra commands table.

12.94.2 Field Documentation

12.94.2.1 `BaseSequentialStream* ShellConfig::sc_channel`

I/O channel associated to the shell.

12.94.2.2 `const ShellCommand* ShellConfig::sc_commands`

Shell extra commands table.

12.95 SPIConfig Struct Reference

12.95.1 Detailed Description

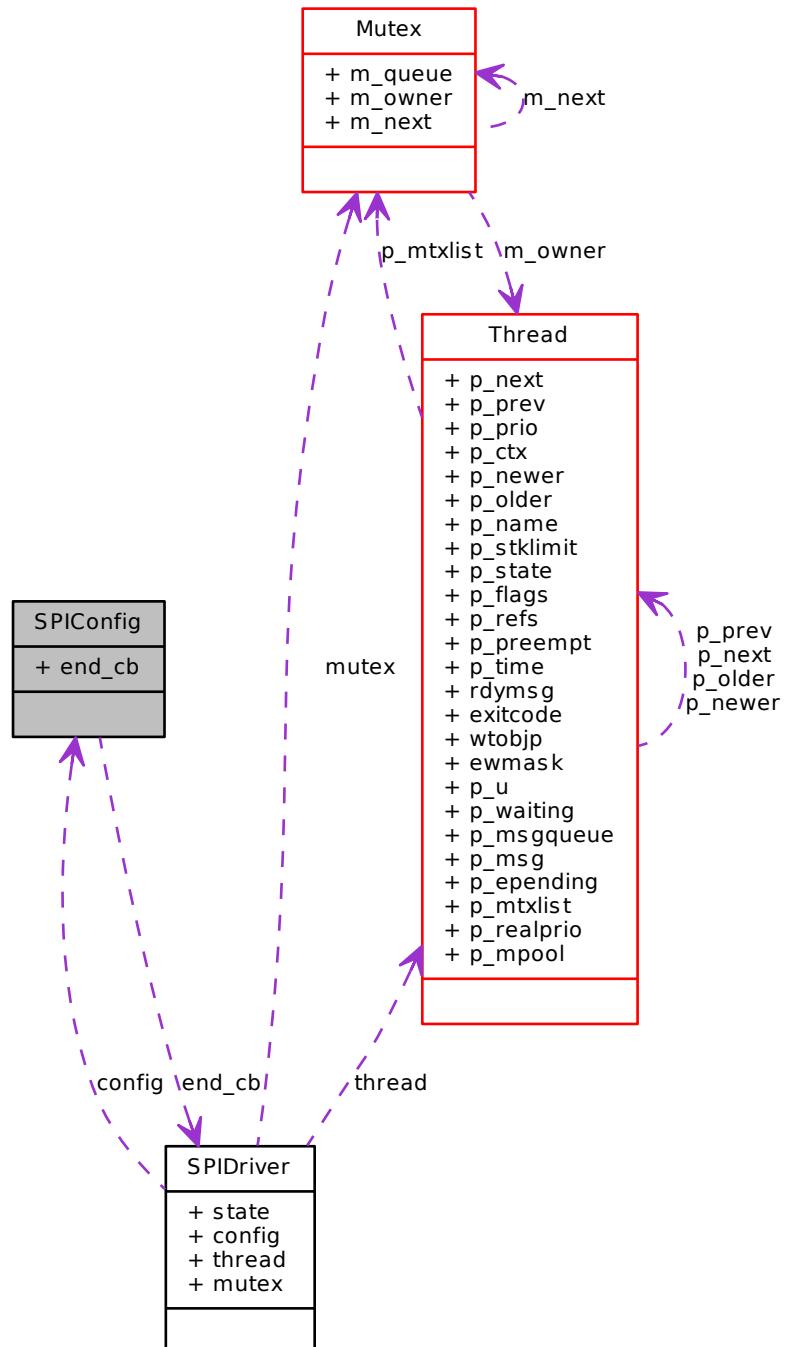
Driver configuration structure.

Note

Implementations may extend this structure to contain more, architecture dependent, fields.

```
#include <spi_lld.h>
```

Collaboration diagram for SPIConfig:



Data Fields

- `spicallback_t end_cb`
Operation complete callback.

12.95.2 Field Documentation

12.95.2.1 spicallback_t SPIConfig::end_cb

Operation complete callback.

12.96 SPIDriver Struct Reference

12.96.1 Detailed Description

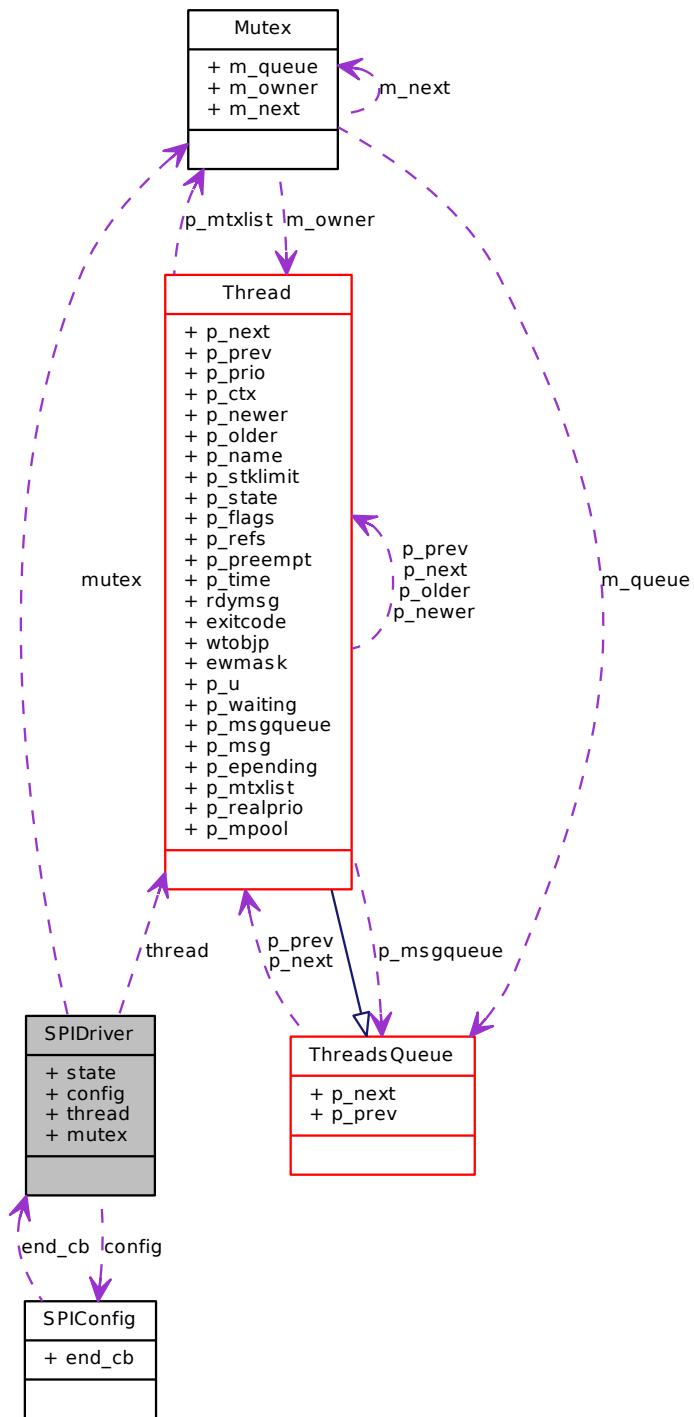
Structure representing an SPI driver.

Note

Implementations may extend this structure to contain more, architecture dependent, fields.

```
#include <spi_llld.h>
```

Collaboration diagram for SPIDriver:



Data Fields

- spistate t state

Driver state

- const **SPIConfig** * config
Current configuration data.
- **Thread** * thread
Waiting thread.
- **Mutex** mutex
Mutex protecting the bus.

12.96.2 Field Documentation

12.96.2.1 spistate_t SPIDriver::state

Driver state.

12.96.2.2 const SPIConfig* SPIDriver::config

Current configuration data.

12.96.2.3 Thread* SPIDriver::thread

Waiting thread.

12.96.2.4 Mutex SPIDriver::mutex

Mutex protecting the bus.

12.97 chibios_rt::System Class Reference

12.97.1 Detailed Description

Class encapsulating the base system functionalities.

```
#include <ch.hpp>
```

Static Public Member Functions

- static void **init** (void)
ChibiOS/RT initialization.
- static void **lock** (void)
Enters the kernel lock mode.
- static void **unlock** (void)
Leaves the kernel lock mode.
- static void **lockFromIlsr** (void)
Enters the kernel lock mode from within an interrupt handler.
- static void **unlockFromIlsr** (void)
Leaves the kernel lock mode from within an interrupt handler.
- static **systime_t** **getTime** (void)
Returns the system time as system ticks.
- static bool **isTimeWithin** (**systime_t** start, **systime_t** end)
Checks if the current system time is within the specified time window.

12.97.2 Member Function Documentation

12.97.2.1 void chibios_rt::System::init(void) [static]

ChibiOS/RT initialization.

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

Precondition

Interrupts must be still disabled when `chSysInit()` is invoked and are internally enabled.

Postcondition

The main thread is created with priority NORMALPRIO.

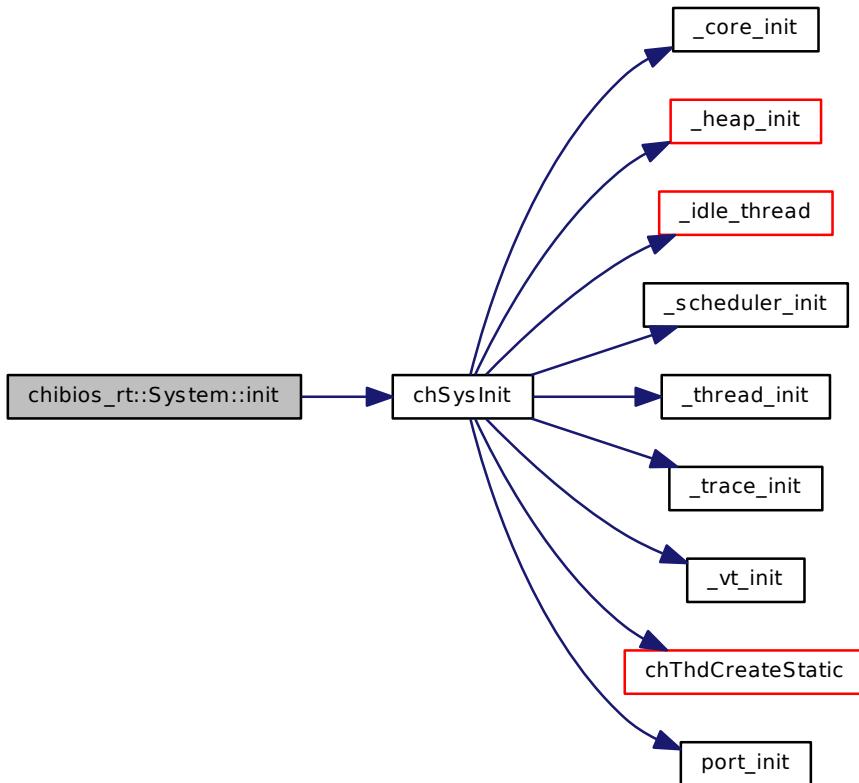
Note

This function has special, architecture-dependent, requirements, see the notes into the various port reference manuals.

Function Class:

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

Here is the call graph for this function:



12.97.2.2 void chibios_rt::System::lock(void) [static]

Enters the kernel lock mode.

Function Class:

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

12.97.2.3 void chibios_rt::System::unlock(void) [static]

Leaves the kernel lock mode.

Function Class:

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

12.97.2.4 void chibios_rt::System::lockFromIsr(void) [static]

Enters the kernel lock mode 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.

12.97.2.5 void chibios_rt::System::unlockFromIsr(void) [static]

Leaves the kernel lock mode 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.

12.97.2.6 systime_t chibios_rt::System::getTime(void) [static]

Returns the system time as system ticks.

Note

The system tick time interval is implementation dependent.

Returns

The system time.

Function Class:

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

12.97.2.7 bool chibios_rt::System::isTimeWithin(systime_t start, systime_t end) [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.

12.98 testcase Struct Reference

12.98.1 Detailed Description

Structure representing a test case.

```
#include <test.h>
```

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.

12.98.2 Field Documentation

12.98.2.1 const char* testcase::name

Test case name.

12.98.2.2 void(* testcase::setup)(void)

Test case preparation function.

12.98.2.3 void(* testcase::teardown)(void)

Test case clean up function.

12.98.2.4 void(* testcase::execute)(void)

Test case execution function.

12.99 Thread Struct Reference

12.99.1 Detailed Description

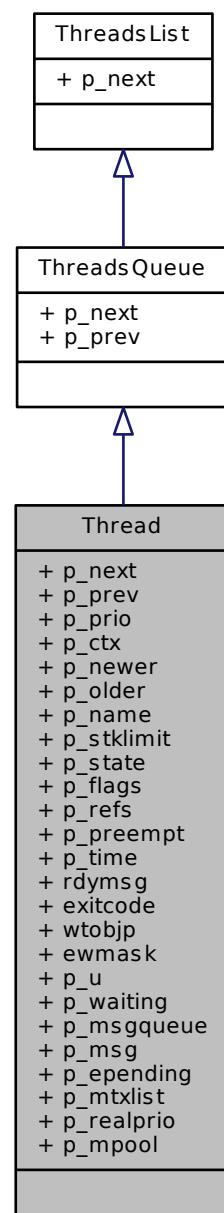
Structure representing a thread.

Note

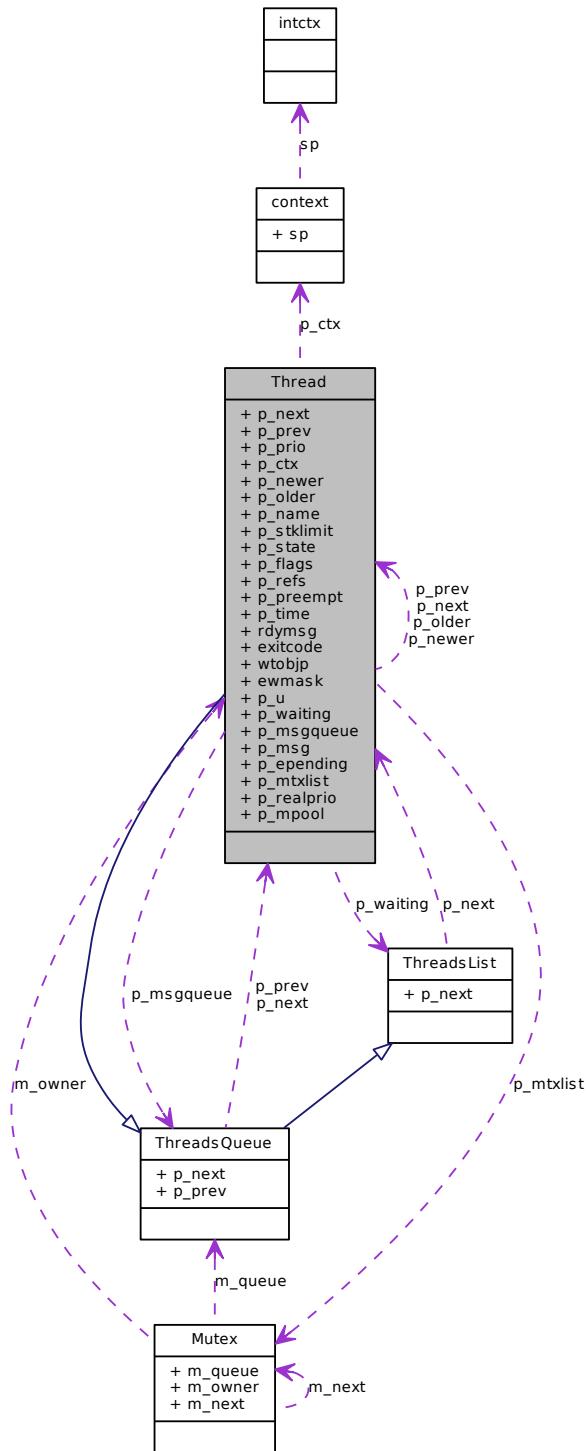
Not all the listed fields are always needed, by switching off some not needed ChibiOS/RT subsystems it is possible to save RAM space by shrinking the [Thread](#) structure.

```
#include <chthreads.h>
```

Inheritance diagram for Thread:



Collaboration diagram for Thread:



Data Fields

- `Thread * p_next`

Next in the list/queue.

- `Thread * p_prev`
Previous in the queue.
- `tprio_t p_prio`
Thread priority.
- `struct context p_ctx`
Processor context.
- `Thread * p_newer`
Newer registry element.
- `Thread * 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.
- `ThreadsList p_waiting`
Termination waiting list.
- `ThreadsQueue p_msgqueue`
Messages queue.
- `msg_t p_msg`
Thread message.
- `eventmask_t p_epending`
Pending events mask.
- `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.
- `msg_t rdymsg`
Thread wakeup code.
- `msg_t exitcode`
Thread exit code.
- `void * wtobjp`
Pointer to a generic "wait" object.
- `eventmask_t ewmask`
Enabled events mask.

12.99.2 Field Documentation

12.99.2.1 `Thread* Thread::p_next`

Next in the list/queue.

Reimplemented from [ThreadsQueue](#).

12.99.2.2 `Thread* Thread::p_prev`

Previous in the queue.

Reimplemented from [ThreadsQueue](#).

12.99.2.3 `tprio_t Thread::p_prio`

[Thread](#) priority.

12.99.2.4 `struct context Thread::p_ctx`

Processor context.

12.99.2.5 `Thread* Thread::p_newer`

Newer registry element.

12.99.2.6 `Thread* Thread::p_older`

Older registry element.

12.99.2.7 `const char* Thread::p_name`

[Thread](#) name or NULL.

12.99.2.8 `stkalign_t* Thread::p_stklimit`

[Thread](#) stack boundary.

12.99.2.9 `tstate_t Thread::p_state`

Current thread state.

12.99.2.10 `tmode_t Thread::p_flags`

Various thread flags.

12.99.2.11 `trefs_t Thread::p_refs`

References to this thread.

12.99.2.12 tslices_t Thread::p_preempt

Number of ticks remaining to this thread.

12.99.2.13 volatile systime_t Thread::p_time

[Thread](#) consumed time in ticks.

Note

This field can overflow.

12.99.2.14 msg_t 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.

12.99.2.15 msg_t 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.

12.99.2.16 void* 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.

12.99.2.17 eventmask_t Thread::ewmask

Enabled events mask.

Note

This field is only valid while the thread is in the THD_STATE_WTOREVT or THD_STATE_WTANDEVT states.

12.99.2.18 ThreadsList Thread::p_waiting

Termination waiting list.

12.99.2.19 ThreadsQueue Thread::p_msgqueue

Messages queue.

12.99.2.20 msg_t Thread::p_msg

[Thread](#) message.

12.99.2.21 eventmask_t Thread::p_epending

Pending events mask.

12.99.2.22 Mutex* Thread::p_mtxlist

List of the mutexes owned by this thread.

Note

The list is terminated by a `NULL` in this field.

12.99.2.23 tprio_t Thread::p_realprio

Thread's own, non-inherited, priority.

12.99.2.24 void* Thread::p_mpool

Memory Pool where the thread workspace is returned.

12.100 chibios_rt::ThreadReference Class Reference

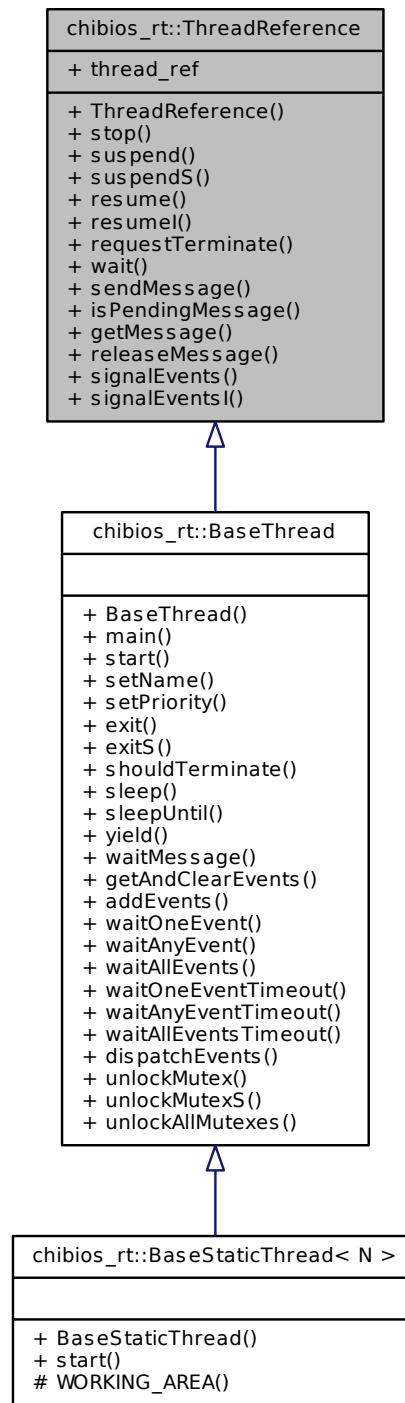
12.100.1 Detailed Description

[Thread](#) reference class.

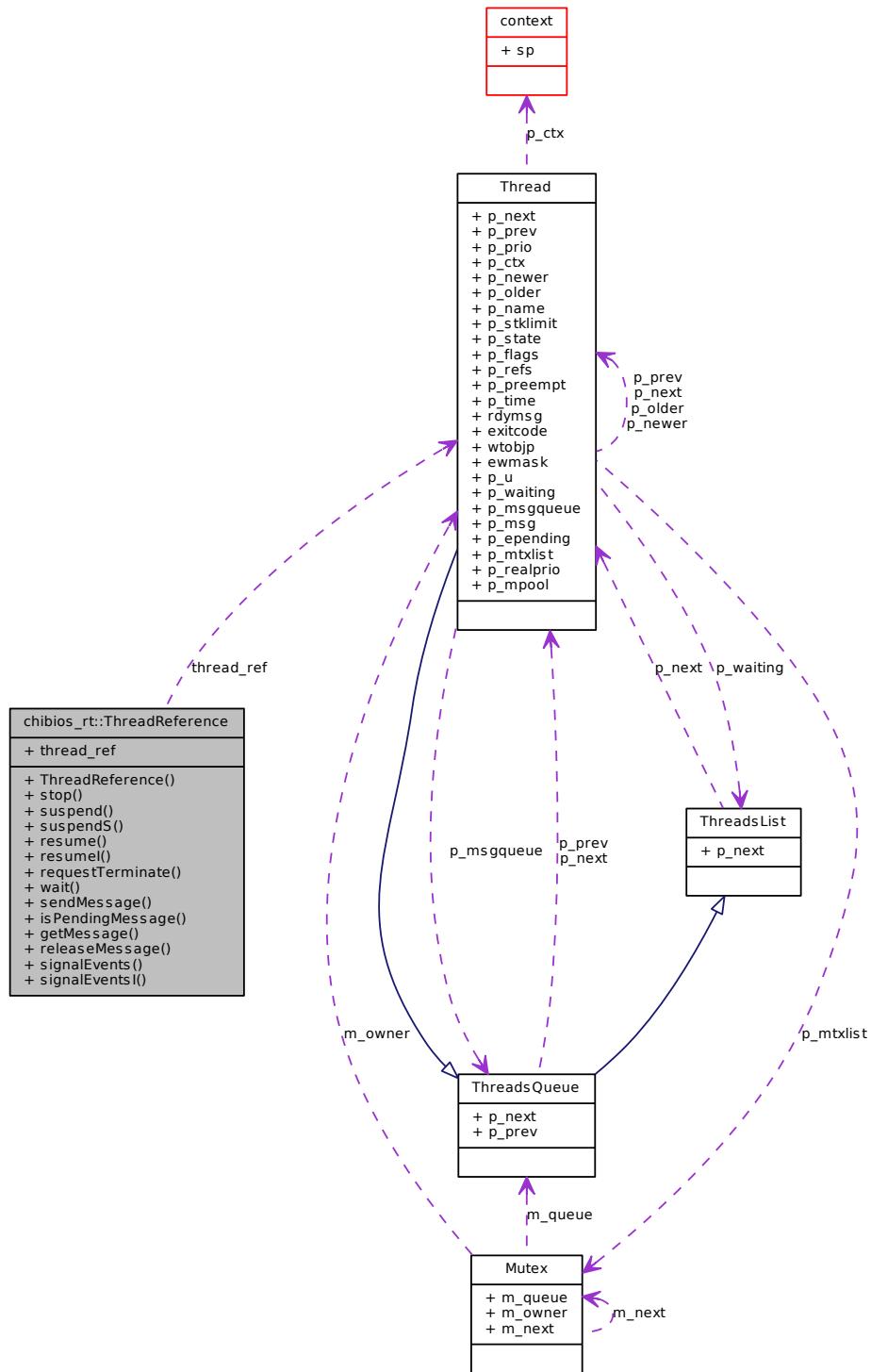
This class encapsulates a reference to a system thread. All operations involving another thread are performed through an object of this type.

```
#include <ch.hpp>
```

Inheritance diagram for chibios_rt::ThreadReference:



Collaboration diagram for chibios_rt::ThreadReference:



Public Member Functions

- **ThreadReference (Thread *tp)**

Thread reference constructor.

- virtual void [stop](#) (void)
Stops the thread.
- [msg_t suspend](#) (void)
Suspends the current thread on the reference.
- [msg_t suspendS](#) (void)
Suspends the current thread on the reference.
- void [resume](#) ([msg_t](#) msg)
Resumes the currently referenced thread, if any.
- void [resumel](#) ([msg_t](#) msg)
Resumes the currently referenced thread, if any.
- void [requestTerminate](#) (void)
Requests a thread termination.
- [msg_t wait](#) (void)
Blocks the execution of the invoking thread until the specified thread terminates then the exit code is returned.
- [msg_t sendMessage](#) ([msg_t](#) msg)
Sends a message to the thread and returns the answer.
- bool [isPendingMessage](#) (void)
Returns true if there is at least one message in queue.
- [msg_t getMessage](#) (void)
Returns an enqueued message or NULL.
- void [releaseMessage](#) ([msg_t](#) msg)
Releases the next message in queue with a reply.
- void [signalEvents](#) ([eventmask_t](#) mask)
Adds a set of event flags directly to specified Thread.
- void [signalEventsI](#) ([eventmask_t](#) mask)
Adds a set of event flags directly to specified Thread.

Data Fields

- [::Thread * thread_ref](#)
Pointer to the system thread.

12.100.2 Constructor & Destructor Documentation

12.100.2.1 chibios_rt::ThreadReference::ThreadReference (Thread * tp) [inline]

[Thread](#) reference constructor.

Parameters

in	<i>tp</i> the target thread. This parameter can be NULL if the thread is not known at creation time.
----	--

Function Class:

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

12.100.3 Member Function Documentation

12.100.3.1 void chibios_rt::ThreadReference::stop (void) [virtual]

Stops the thread.

Note

The implementation is left to descendant classes and is optional.

Here is the call graph for this function:

**12.100.3.2 msg_t chibios_rt::ThreadReference::suspend (void)**

Suspends the current thread on the reference.

The suspended thread becomes the referenced thread. It is possible to use this method only if the thread reference was set to NULL.

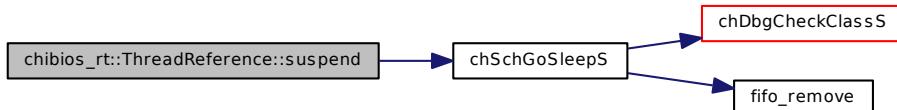
Returns

The incoming 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:

**12.100.3.3 msg_t chibios_rt::ThreadReference::suspendS (void)**

Suspends the current thread on the reference.

The suspended thread becomes the referenced thread. It is possible to use this method only if the thread reference was set to NULL.

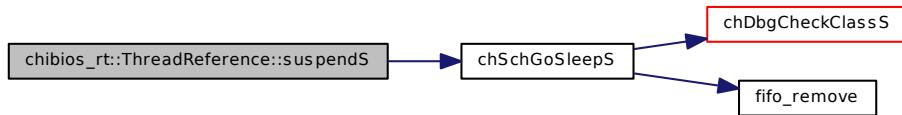
Returns

The incoming 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:



12.100.3.4 void chibios_rt::ThreadReference::resume (msg_t msg)

Resumes the currently referenced thread, if any.

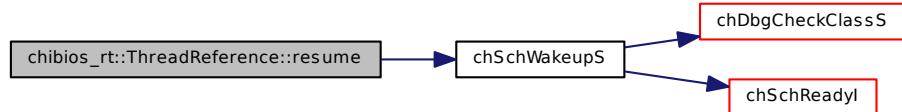
Parameters

in *msg* the wakeup 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:



12.100.3.5 void chibios_rt::ThreadReference::resumel (msg_t msg)

Resumes the currently referenced thread, if any.

Parameters

in *msg* the wakeup message

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:



12.100.3.6 void chibios_rt::ThreadReference::requestTerminate (void)

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.

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:



12.100.3.7 msg_t chibios_rt::ThreadReference::wait (void)

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

Note

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

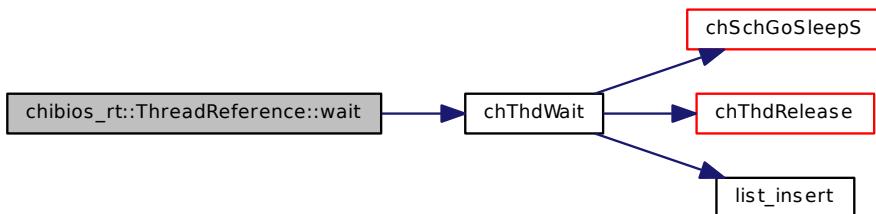
Returns

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:

**12.100.3.8 msg_t chibios_rt::ThreadReference::sendMessage(msg_t msg)**

Sends a message to the thread and returns the answer.

Parameters

in *msg* the sent message

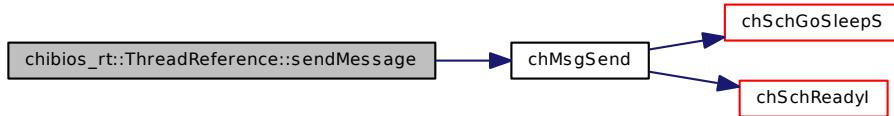
Returns

The returned 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:



12.100.3.9 bool chibios_rt::ThreadReference::isPendingMessage (void)

Returns true if there is at least one message in queue.

Return values

<i>true</i>	A message is waiting in queue.
<i>false</i>	A message is not waiting in queue.

Function Class:

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

12.100.3.10 msg_t chibios_rt::ThreadReference::getMessage (void)

Returns an enqueued message or NULL.

Returns

The incoming message.

Function Class:

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

12.100.3.11 void chibios_rt::ThreadReference::releaseMessage (msg_t msg)

Releases the next message in queue with a reply.

Parameters

<i>in</i>	<i>msg</i> the answer 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:



12.100.3.12 void chibios_rt::ThreadReference::signalEvents (eventmask_t mask)

Adds a set of event flags directly to specified [Thread](#).

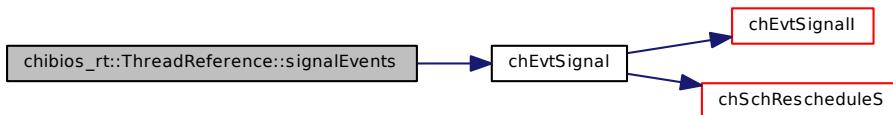
Parameters

in *mask* the event flags 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:



12.100.3.13 void chibios_rt::ThreadReference::signalEventsI (eventmask_t mask)

Adds a set of event flags directly to specified [Thread](#).

Parameters

in *mask* the event flags 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:



12.100.4 Field Documentation

12.100.4.1 ::Thread* chibios_rt::ThreadReference::thread_ref

Pointer to the system thread.

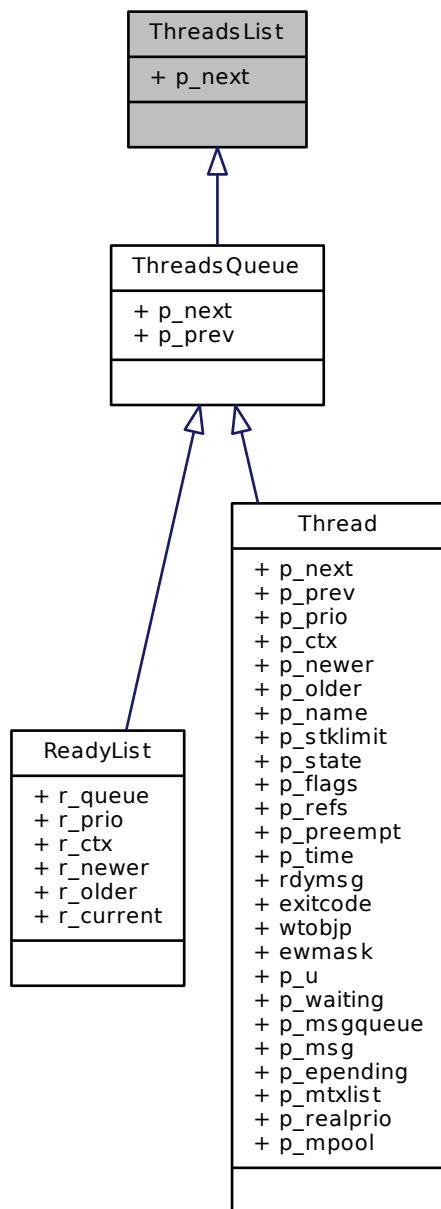
12.101 ThreadsList Struct Reference

12.101.1 Detailed Description

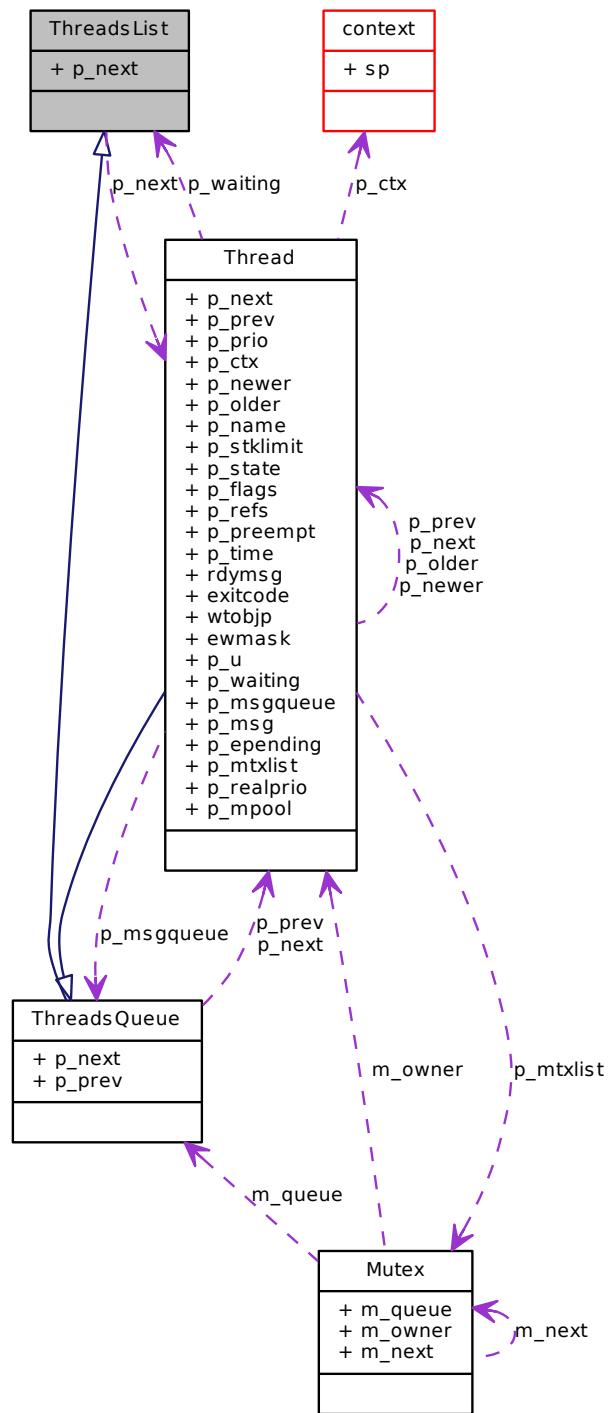
Generic threads single link list, it works like a stack.

```
#include <chlists.h>
```

Inheritance diagram for ThreadsList:



Collaboration diagram for ThreadsList:



Data Fields

- `Thread * p_next`

12.101.2 Field Documentation

12.101.2.1 Thread* ThreadsList::p_next

Last pushed [Thread](#) on the stack list, or pointer to itself if empty.

Reimplemented in [ThreadsQueue](#), and [Thread](#).

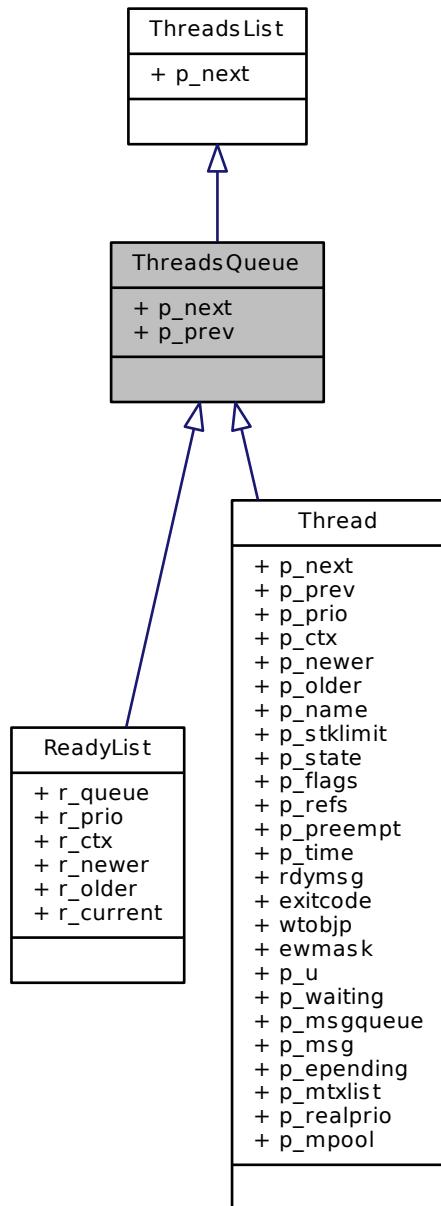
12.102 ThreadsQueue Struct Reference

12.102.1 Detailed Description

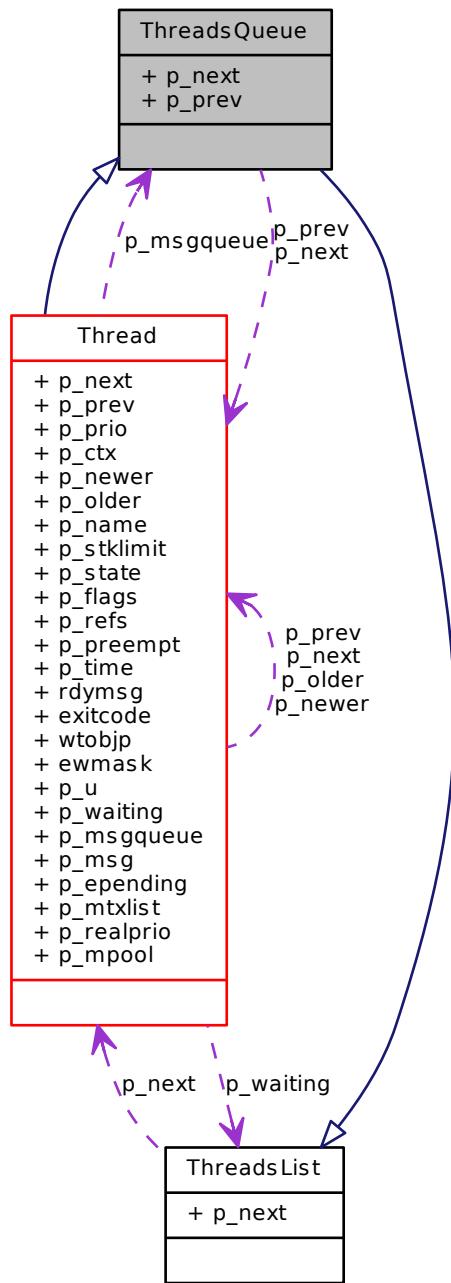
Generic threads bidirectional linked list header and element.

```
#include <chlists.h>
```

Inheritance diagram for ThreadsQueue:



Collaboration diagram for ThreadsQueue:



Data Fields

- `Thread * p_next`
- `Thread * p_prev`

12.102.2 Field Documentation

12.102.2.1 Thread* ThreadsQueue::p_next

First [Thread](#) in the queue, or [ThreadQueue](#) when empty.

Reimplemented from [ThreadsList](#).

Reimplemented in [Thread](#).

12.102.2.2 Thread* ThreadsQueue::p_prev

Last [Thread](#) in the queue, or [ThreadQueue](#) when empty.

Reimplemented in [Thread](#).

12.103 TimeMeasurement Struct Reference

12.103.1 Detailed Description

Time Measurement structure.

```
#include <tm.h>
```

Data Fields

- `void(* start)(TimeMeasurement *tmp)`
Starts a measurement.
- `void(* stop)(TimeMeasurement *tmp)`
Stops a measurement.
- `halrcnt_t last`
Last measurement.
- `halrcnt_t worst`
Worst measurement.
- `halrcnt_t best`
Best measurement.

12.103.2 Field Documentation

12.103.2.1 void(* TimeMeasurement::start)(TimeMeasurement *tmp)

Starts a measurement.

12.103.2.2 void(* TimeMeasurement::stop)(TimeMeasurement *tmp)

Stops a measurement.

12.103.2.3 halrcnt_t TimeMeasurement::last

Last measurement.

12.103.2.4 halrcnt_t TimeMeasurement::worst

Worst measurement.

12.103.2.5 halrcnt_t TimeMeasurement::best

Best measurement.

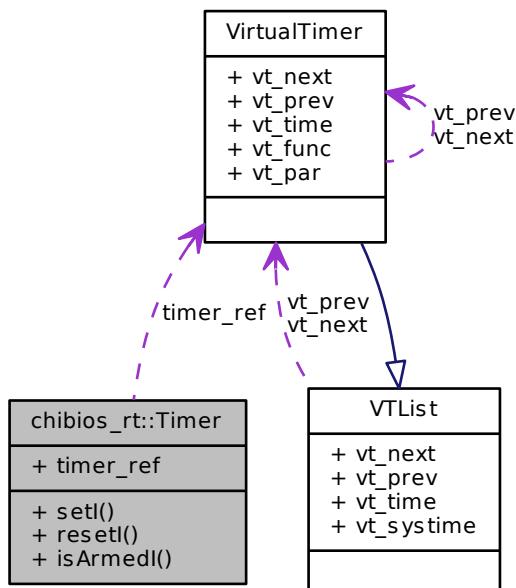
12.104 chibios_rt::Timer Class Reference

12.104.1 Detailed Description

[Timer](#) class.

```
#include <ch.hpp>
```

Collaboration diagram for chibios_rt::Timer:



Public Member Functions

- void `setl (systime_t time, vfunc_t vtfunc, void *par)`
Enables a virtual timer.
- void `resetl ()`
Resets the timer, if armed.
- bool `isArmedl (void)`
Returns the timer status.

Data Fields

- ::VirtualTimer `timer_ref`
Embedded `VirtualTimer` structure.

12.104.2 Member Function Documentation

12.104.2.1 void chibios_rt::Timer::setl (systime_t time, vfunc_t vfunc, void * par)

Enables a virtual timer.

Note

The associated function is invoked from interrupt context.

Parameters

in *time* the number of ticks before the operation timeouts, the special values are handled as follow:

- *TIME_INFINITE* is allowed but interpreted as a normal time specification.
- *TIME_IMMEDIATE* this value is not allowed.

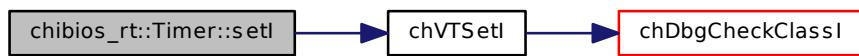
in *vfunc* the timer callback function. After invoking the callback the timer is disabled and the structure can be disposed or reused.

in *par* 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:



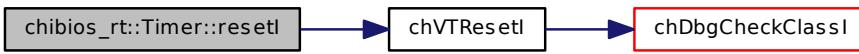
12.104.2.2 void chibios_rt::Timer::resetl ()

Resets the timer, if 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:



12.104.2.3 bool chibios_rt::Timer::isArmed() void)

Returns the timer status.

Return values

<i>TRUE</i>	The timer is armed.
<i>FALSE</i>	The timer already fired its callback.

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.

12.104.3 Field Documentation

12.104.3.1 ::VirtualTimer chibios_rt::Timer::timer_ref

Embedded [VirtualTimer](#) structure.

12.105 UARTConfig Struct Reference

12.105.1 Detailed Description

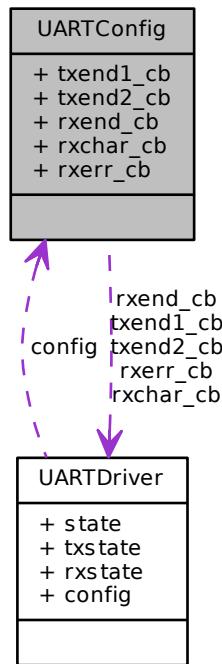
Driver configuration structure.

Note

Implementations may extend this structure to contain more, architecture dependent, fields.

```
#include <uart_lld.h>
```

Collaboration diagram for UARTConfig:



Data Fields

- `uartcb_t txend1_cb`
End of transmission buffer callback.
- `uartcb_t txend2_cb`
Physical end of transmission callback.
- `uartcb_t rxend_cb`
Receive buffer filled callback.
- `uartccb_t rxchar_cb`
Character received while out if the `UART_RECEIVE` state.
- `uartecb_t rxerr_cb`
Receive error callback.

12.105.2 Field Documentation

12.105.2.1 `uartcb_t UARTConfig::txend1_cb`

End of transmission buffer callback.

12.105.2.2 `uartcb_t UARTConfig::txend2_cb`

Physical end of transmission callback.

12.105.2.3 `uartcb_t` `UARTConfig::rxend_cb`

Receive buffer filled callback.

12.105.2.4 `uartccb_t` `UARTConfig::rxchar_cb`

Character received while out if the `UART_RECEIVE` state.

12.105.2.5 `uartecb_t` `UARTConfig::rxerr_cb`

Receive error callback.

12.106 UARTDriver Struct Reference

12.106.1 Detailed Description

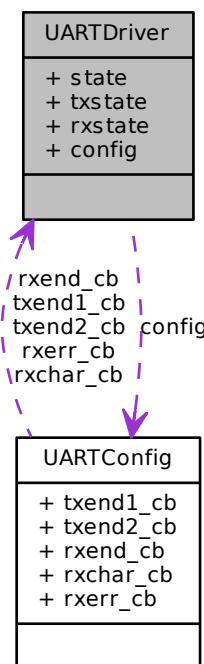
Structure representing an UART driver.

Note

Implementations may extend this structure to contain more, architecture dependent, fields.

```
#include <uart_lld.h>
```

Collaboration diagram for `UARTDriver`:



Data Fields

- `uartstate_t state`
Driver state.
- `uarttxstate_t txstate`
Transmitter state.
- `uartrxstate_t rxstate`
Receiver state.
- const `UARTConfig * config`
Current configuration data.

12.106.2 Field Documentation

12.106.2.1 `uartstate_t UARTDriver::state`

Driver state.

12.106.2.2 `uarttxstate_t UARTDriver::txstate`

Transmitter state.

12.106.2.3 `uartrxstate_t UARTDriver::rxstate`

Receiver state.

12.106.2.4 const `UARTConfig* UARTDriver::config`

Current configuration data.

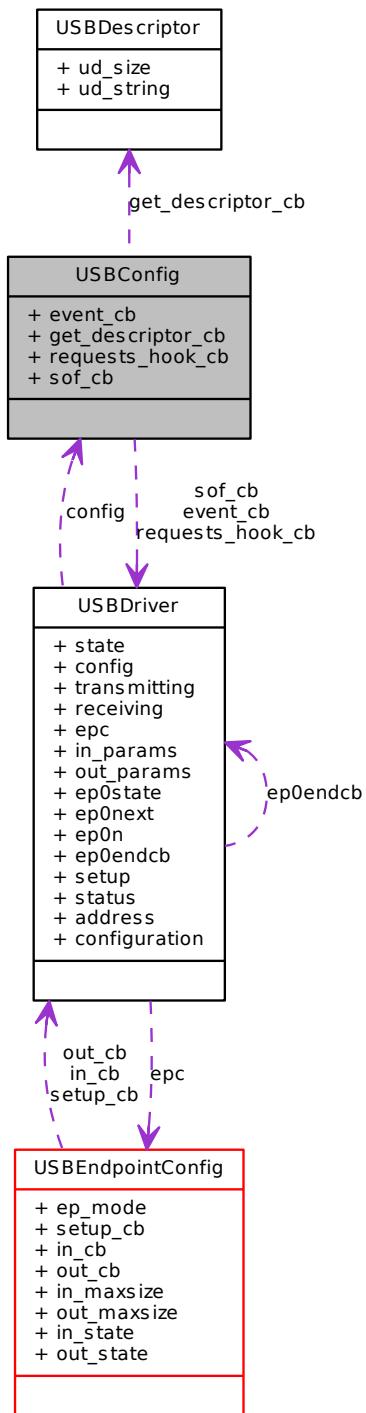
12.107 USBConfig Struct Reference

12.107.1 Detailed Description

Type of an USB driver configuration structure.

```
#include <usb_lld.h>
```

Collaboration diagram for USBConfig:



Data Fields

- `usbeventcb_t event_cb`

USB events callback.

- `usbgetdescriptor_t get_descriptor_cb`
Device GET_DESCRIPTOR request callback.
- `usbreqhandler_t requests_hook_cb`
Requests hook callback.
- `usbcallback_t sof_cb`
Start Of Frame callback.

12.107.2 Field Documentation

12.107.2.1 usbeventcb_t USBConfig::event_cb

USB events callback.

This callback is invoked when an USB driver event is registered.

12.107.2.2 usbgetdescriptor_t USBConfig::get_descriptor_cb

Device GET_DESCRIPTOR request callback.

Note

This callback is mandatory and cannot be set to NULL.

12.107.2.3 usbreqhandler_t USBConfig::requests_hook_cb

Requests hook callback.

This hook allows to be notified of standard requests or to handle non standard requests.

12.107.2.4 usbcallback_t USBConfig::sof_cb

Start Of Frame callback.

12.108 USBDescriptor Struct Reference

12.108.1 Detailed Description

Type of an USB descriptor.

```
#include <usb.h>
```

Data Fields

- `size_t ud_size`
Descriptor size in unicode characters.
- `const uint8_t * ud_string`
Pointer to the descriptor.

12.108.2 Field Documentation

12.108.2.1 size_t USBDescriptor::ud_size

Descriptor size in unicode characters.

12.108.2.2 const uint8_t* USBDescriptor::ud_string

Pointer to the descriptor.

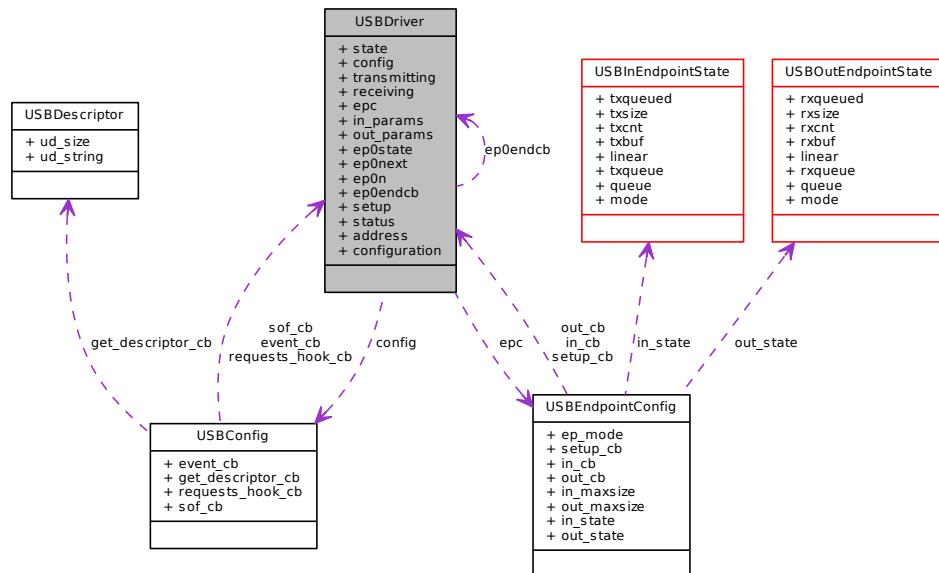
12.109 USBDriver Struct Reference

12.109.1 Detailed Description

Structure representing an USB driver.

```
#include <usb_lld.h>
```

Collaboration diagram for USBDriver:



Data Fields

- **usbstate_t state**
Driver state.
- **const USBConfig * config**
Current configuration data.
- **uint16_t transmitting**
Bit map of the transmitting IN endpoints.
- **uint16_t receiving**
Bit map of the receiving OUT endpoints.
- **const USBEndpointConfig * epc [USB_MAX_ENDPOINTS+1]**
Active endpoints configurations.
- **void * in_params [USB_MAX_ENDPOINTS]**
Fields available to user, it can be used to associate an application-defined handler to an IN endpoint.
- **void * out_params [USB_MAX_ENDPOINTS]**
Fields available to user, it can be used to associate an application-defined handler to an OUT endpoint.
- **usbep0state_t ep0state**

- *Endpoint 0 state.*
- `uint8_t * ep0next`
Next position in the buffer to be transferred through endpoint 0.
- `size_t ep0n`
Number of bytes yet to be transferred through endpoint 0.
- `usbcallback_t ep0endcb`
Endpoint 0 end transaction callback.
- `uint8_t setup [8]`
Setup packet buffer.
- `uint16_t status`
Current USB device status.
- `uint8_t address`
Assigned USB address.
- `uint8_t configuration`
Current USB device configuration.

12.109.2 Field Documentation

12.109.2.1 `usbstate_t USBDriver::state`

Driver state.

12.109.2.2 `const USBConfig* USBDriver::config`

Current configuration data.

12.109.2.3 `uint16_t USBDriver::transmitting`

Bit map of the transmitting IN endpoints.

12.109.2.4 `uint16_t USBDriver::receiving`

Bit map of the receiving OUT endpoints.

12.109.2.5 `const USBEndpointConfig* USBDriver::epc[USB_MAX_ENDPOINTS+1]`

Active endpoints configurations.

12.109.2.6 `void* USBDriver::in_params[USB_MAX_ENDPOINTS]`

Fields available to user, it can be used to associate an application-defined handler to an IN endpoint.

Note

The base index is one, the endpoint zero does not have a reserved element in this array.

12.109.2.7 `void* USBDriver::out_params[USB_MAX_ENDPOINTS]`

Fields available to user, it can be used to associate an application-defined handler to an OUT endpoint.

Note

The base index is one, the endpoint zero does not have a reserved element in this array.

12.109.2.8 `usbep0state_t` **USBDriver::ep0state**

Endpoint 0 state.

12.109.2.9 `uint8_t*` **USBDriver::ep0next**

Next position in the buffer to be transferred through endpoint 0.

12.109.2.10 `size_t` **USBDriver::ep0n**

Number of bytes yet to be transferred through endpoint 0.

12.109.2.11 `usbcallback_t` **USBDriver::ep0endcb**

Endpoint 0 end transaction callback.

12.109.2.12 `uint8_t` **USBDriver::setup[8]**

Setup packet buffer.

12.109.2.13 `uint16_t` **USBDriver::status**

Current USB device status.

12.109.2.14 `uint8_t` **USBDriver::address**

Assigned USB address.

12.109.2.15 `uint8_t` **USBDriver::configuration**

Current USB device configuration.

12.110 USBEndpointConfig Struct Reference

12.110.1 Detailed Description

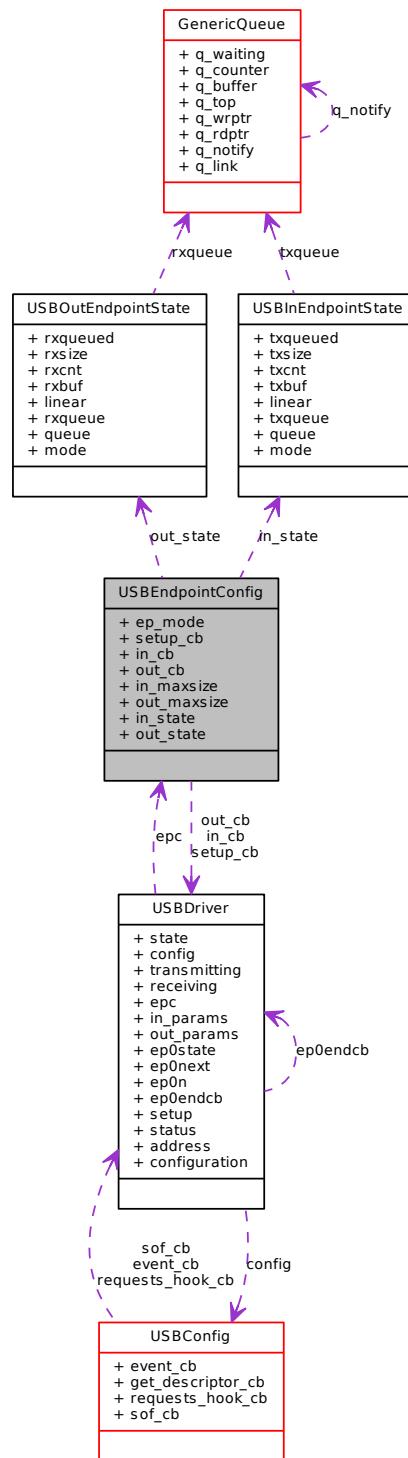
Type of an USB endpoint configuration structure.

Note

Platform specific restrictions may apply to endpoints.

```
#include <usb_lld.h>
```

Collaboration diagram for USBEndpointConfig:



Data Fields

- `uint32_t ep_mode`

Type and mode of the endpoint.

- `usbepcallback_t setup_cb`
Setup packet notification callback.
- `usbepcallback_t in_cb`
IN endpoint notification callback.
- `usbepcallback_t out_cb`
OUT endpoint notification callback.
- `uint16_t in_maxsize`
IN endpoint maximum packet size.
- `uint16_t out_maxsize`
OUT endpoint maximum packet size.
- `USBInEndpointState * in_state`
USBInEndpointState associated to the IN endpoint.
- `USBOutEndpointState * out_state`
USBOutEndpointState associated to the OUT endpoint.

12.110.2 Field Documentation

12.110.2.1 `uint32_t USBEndpointConfig::ep_mode`

Type and mode of the endpoint.

12.110.2.2 `usbepcallback_t USBEndpointConfig::setup_cb`

Setup packet notification callback.

This callback is invoked when a setup packet has been received.

Postcondition

The application must immediately call `usbReadPacket()` in order to access the received packet.

Note

This field is only valid for `USB_EP_MODE_TYPE_CTRL` endpoints, it should be set to `NULL` for other endpoint types.

12.110.2.3 `usbepcallback_t USBEndpointConfig::in_cb`

IN endpoint notification callback.

This field must be set to `NULL` if the IN endpoint is not used.

12.110.2.4 `usbepcallback_t USBEndpointConfig::out_cb`

OUT endpoint notification callback.

This field must be set to `NULL` if the OUT endpoint is not used.

12.110.2.5 `uint16_t USBEndpointConfig::in_maxsize`

IN endpoint maximum packet size.

This field must be set to zero if the IN endpoint is not used.

12.110.2.6 uint16_t USBEndpointConfig::out_maxsize

OUT endpoint maximum packet size.

This field must be set to zero if the OUT endpoint is not used.

12.110.2.7 USBInEndpointState* USBEndpointConfig::in_state

USBEndpointState associated to the IN endpoint.

This structure maintains the state of the IN endpoint.

12.110.2.8 USBOutEndpointState* USBEndpointConfig::out_state

USBEndpointState associated to the OUT endpoint.

This structure maintains the state of the OUT endpoint.

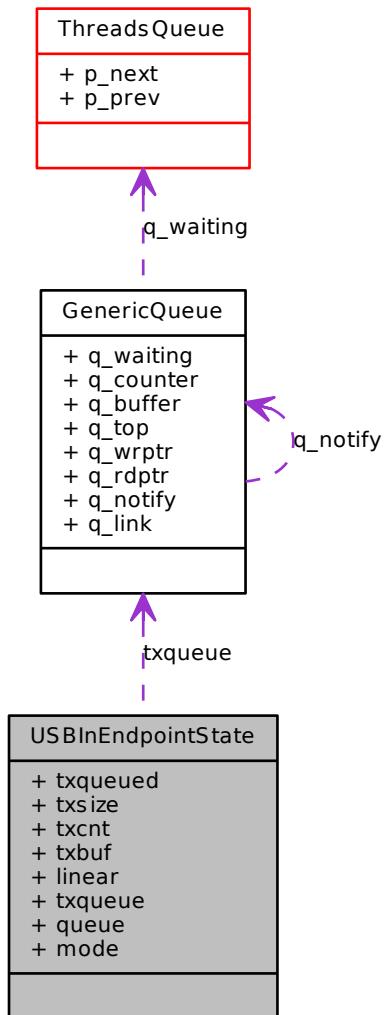
12.111 USBInEndpointState Struct Reference

12.111.1 Detailed Description

Type of an IN endpoint state structure.

```
#include <usb_lld.h>
```

Collaboration diagram for USBInEndpointState:



Data Fields

- `bool_t txqueued`
Buffer mode, queue or linear.
- `size_t txsize`
Requested transmit transfer size.
- `size_t txcnt`
Transmitted bytes so far.
- `const uint8_t * txbuf`
Pointer to the transmission linear buffer.
- `OutputQueue * txqueue`
Pointer to the output queue.

12.111.2 Field Documentation

12.111.2.1 `bool_t USBInEndpointState::txqueued`

Buffer mode, queue or linear.

12.111.2.2 `size_t USBInEndpointState::txsize`

Requested transmit transfer size.

12.111.2.3 `size_t USBInEndpointState::txcnt`

Transmitted bytes so far.

12.111.2.4 `const uint8_t* USBInEndpointState::txbuf`

Pointer to the transmission linear buffer.

12.111.2.5 `OutputQueue* USBInEndpointState::txqueue`

Pointer to the output queue.

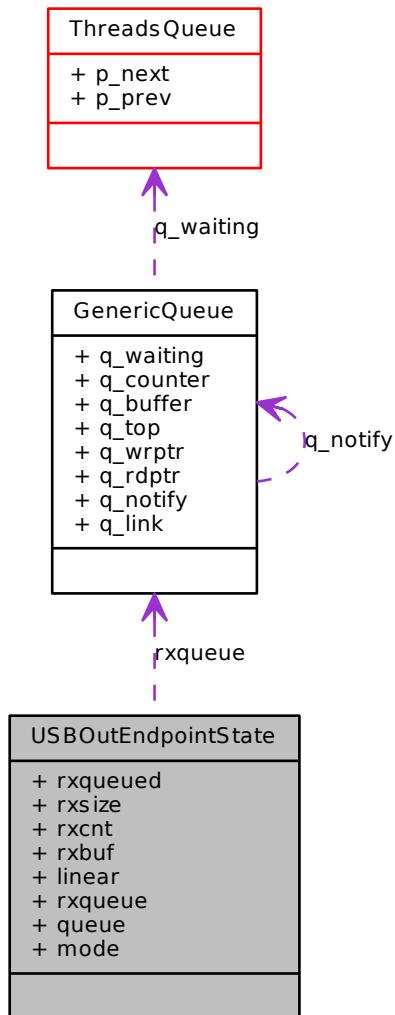
12.112 USBOutEndpointState Struct Reference

12.112.1 Detailed Description

Type of an OUT endpoint state structure.

```
#include <usb_lld.h>
```

Collaboration diagram for USBOutEndpointState:



Data Fields

- `bool_t rxqueued`
Buffer mode, queue or linear.
- `size_t rxszie`
Requested receive transfer size.
- `size_t rxcnt`
Received bytes so far.
- `uint8_t * rdbuf`
Pointer to the receive linear buffer.
- `InputQueue * rxqueue`
Pointer to the input queue.

12.112.2 Field Documentation

12.112.2.1 `bool_t USBOutEndpointState::rxqueued`

Buffer mode, queue or linear.

12.112.2.2 `size_t USBOutEndpointState::rxsize`

Requested receive transfer size.

12.112.2.3 `size_t USBOutEndpointState::rxcnt`

Received bytes so far.

12.112.2.4 `uint8_t* USBOutEndpointState::rxbuf`

Pointer to the receive linear buffer.

12.112.2.5 `InputQueue* USBOutEndpointState::rxqueue`

Pointer to the input queue.

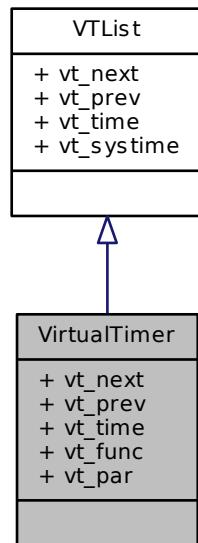
12.113 VirtualTimer Struct Reference

12.113.1 Detailed Description

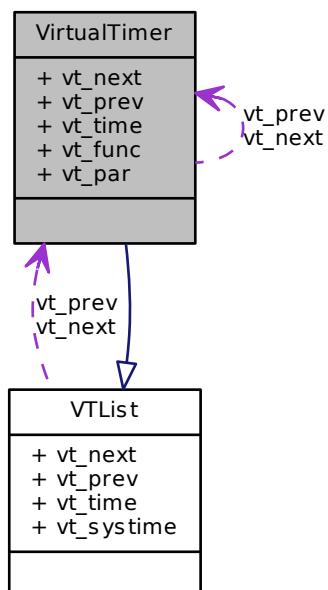
Virtual Timer descriptor structure.

```
#include <chvt.h>
```

Inheritance diagram for VirtualTimer:



Collaboration diagram for VirtualTimer:



Data Fields

- `VirtualTimer * vt_next`
Next timer in the delta list.
- `VirtualTimer * vt_prev`
Previous timer in the delta list.
- `systime_t vt_time`
Time delta before timeout.
- `vfunc_t vt_func`
Timer callback function pointer.
- `void * vt_par`
Timer callback function parameter.

12.113.2 Field Documentation

12.113.2.1 `VirtualTimer* VirtualTimer::vt_next`

Next timer in the delta list.

Reimplemented from [VTList](#).

12.113.2.2 `VirtualTimer* VirtualTimer::vt_prev`

Previous timer in the delta list.

Reimplemented from [VTList](#).

12.113.2.3 `systime_t VirtualTimer::vt_time`

Time delta before timeout.

Reimplemented from [VTList](#).

12.113.2.4 `vfunc_t VirtualTimer::vt_func`

Timer callback function pointer.

12.113.2.5 `void* VirtualTimer::vt_par`

Timer callback function parameter.

12.114 VTList Struct Reference

12.114.1 Detailed Description

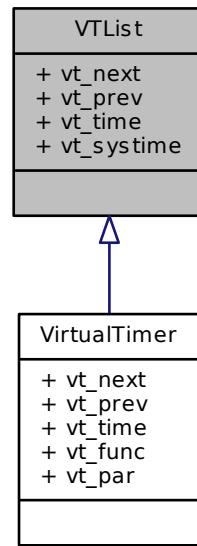
Virtual timers list header.

Note

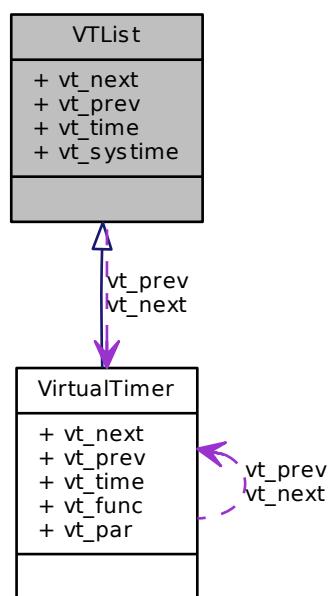
The delta list is implemented as a double link bidirectional list in order to make the unlink time constant, the reset of a virtual timer is often used in the code.

```
#include <chvt.h>
```

Inheritance diagram for VTList:



Collaboration diagram for VTList:



Data Fields

- `VirtualTimer * vt_next`
Next timer in the delta list.
- `VirtualTimer * vt_prev`
Last timer in the delta list.
- `systime_t vt_time`
Must be initialized to -1.
- `volatile systime_t vt_systime`
System Time counter.

12.114.2 Field Documentation

12.114.2.1 `VirtualTimer* VList::vt_next`

Next timer in the delta list.

Reimplemented in [VirtualTimer](#).

12.114.2.2 `VirtualTimer* VList::vt_prev`

Last timer in the delta list.

Reimplemented in [VirtualTimer](#).

12.114.2.3 `systime_t VList::vt_time`

Must be initialized to -1.

Reimplemented in [VirtualTimer](#).

12.114.2.4 `volatile systime_t VList::vt_systime`

System Time counter.

Chapter 13

File Documentation

13.1 adc.c File Reference

13.1.1 Detailed Description

ADC Driver code.

```
#include "ch.h"
#include "hal.h"
```

Functions

- void `adclInit` (void)
ADC Driver initialization.
- void `adcObjectInit` (`ADCDriver` *adcp)
Initializes the standard part of a `ADCDriver` structure.
- void `adcStart` (`ADCDriver` *adcp, const `ADCConfig` *config)
Configures and activates the ADC peripheral.
- void `adcStop` (`ADCDriver` *adcp)
Deactivates the ADC peripheral.
- void `adcStartConversion` (`ADCDriver` *adcp, const `ADCConversionGroup` *grpp, `adcsample_t` *samples, `size_t` depth)
Starts an ADC conversion.
- void `adcStartConversionl` (`ADCDriver` *adcp, const `ADCConversionGroup` *grpp, `adcsample_t` *samples, `size_t` depth)
Starts an ADC conversion.
- void `adcStopConversion` (`ADCDriver` *adcp)
Stops an ongoing conversion.
- void `adcStopConversionl` (`ADCDriver` *adcp)
Stops an ongoing conversion.
- msg_t `adcConvert` (`ADCDriver` *adcp, const `ADCConversionGroup` *grpp, `adcsample_t` *samples, `size_t` depth)
Performs an ADC conversion.
- void `adcAcquireBus` (`ADCDriver` *adcp)
Gains exclusive access to the ADC peripheral.
- void `adcReleaseBus` (`ADCDriver` *adcp)
Releases exclusive access to the ADC peripheral.

13.2 adc.h File Reference

13.2.1 Detailed Description

ADC Driver macros and structures. #include "adc_lld.h"

Functions

- void **adclinit** (void)
ADC Driver initialization.
- void **adcObjectInit** (ADCDriver *adcp)
Initializes the standard part of a `ADCDriver` structure.
- void **adcStart** (ADCDriver *adcp, const ADCConfig *config)
Configures and activates the ADC peripheral.
- void **adcStop** (ADCDriver *adcp)
Deactivates the ADC peripheral.
- void **adcStartConversion** (ADCDriver *adcp, const ADCConversionGroup *grpp, adcsample_t *samples, size_t depth)
Starts an ADC conversion.
- void **adcStartConversionI** (ADCDriver *adcp, const ADCConversionGroup *grpp, adcsample_t *samples, size_t depth)
Starts an ADC conversion.
- void **adcStopConversion** (ADCDriver *adcp)
Stops an ongoing conversion.
- void **adcStopConversionI** (ADCDriver *adcp)
Stops an ongoing conversion.
- void **adcAcquireBus** (ADCDriver *adcp)
Gains exclusive access to the ADC peripheral.
- void **adcReleaseBus** (ADCDriver *adcp)
Releases exclusive access to the ADC peripheral.

Defines

ADC configuration options

- #define **ADC_USE_WAIT** TRUE
Enables synchronous APIs.
- #define **ADC_USE_MUTUAL_EXCLUSION** TRUE
Enables the `adcAcquireBus()` and `adcReleaseBus()` APIs.

Low Level driver helper macros

- #define **_adc_reset_i**(adcp)
Resumes a thread waiting for a conversion completion.
- #define **_adc_reset_s**(adcp)
Resumes a thread waiting for a conversion completion.
- #define **_adc_wakeup_isr**(adcp)
Wakes up the waiting thread.
- #define **_adc_timeout_isr**(adcp)
Wakes up the waiting thread with a timeout message.
- #define **_adc_isr_half_code**(adcp)
Common ISR code, half buffer event.
- #define **_adc_isr_full_code**(adcp)
Common ISR code, full buffer event.
- #define **_adc_isr_error_code**(adcp, err)
Common ISR code, error event.

Enumerations

- enum `adcstate_t` {
 `ADC_UNINIT` = 0, `ADC_STOP` = 1, `ADC_READY` = 2, `ADC_ACTIVE` = 3,
 `ADC_COMPLETE` = 4, `ADC_ERROR` = 5 }
Driver state machine possible states.

13.3 adc_lld.c File Reference

13.3.1 Detailed Description

ADC Driver subsystem low level driver source template.

```
#include "ch.h"  
#include "hal.h"
```

Functions

- void `adc_lld_init` (void)
Low level ADC driver initialization.
- void `adc_lld_start` (ADCDriver *adcp)
Configures and activates the ADC peripheral.
- void `adc_lld_stop` (ADCDriver *adcp)
Deactivates the ADC peripheral.
- void `adc_lld_start_conversion` (ADCDriver *adcp)
Starts an ADC conversion.
- void `adc_lld_stop_conversion` (ADCDriver *adcp)
Stops an ongoing conversion.

Variables

- ADCDriver `ADCD1`
ADC1 driver identifier.

13.4 adc_lld.h File Reference

13.4.1 Detailed Description

ADC Driver subsystem low level driver header template.

Data Structures

- struct `ADCCConversionGroup`
Conversion group configuration structure.
- struct `ADCCConfig`
Driver configuration structure.
- struct `ADCDriver`
Structure representing an ADC driver.

Functions

- void `adc_lld_init` (void)
Low level ADC driver initialization.
- void `adc_lld_start` (`ADCDriver` *adcp)
Configures and activates the ADC peripheral.
- void `adc_lld_stop` (`ADCDriver` *adcp)
Deactivates the ADC peripheral.
- void `adc_lld_start_conversion` (`ADCDriver` *adcp)
Starts an ADC conversion.
- void `adc_lld_stop_conversion` (`ADCDriver` *adcp)
Stops an ongoing conversion.

Defines

Configuration options

- `#define PLATFORM_ADC_USE_ADC1 FALSE`
ADC1 driver enable switch.

Typedefs

- `typedef uint16_t adcsample_t`
ADC sample data type.
- `typedef uint16_t adc_channels_num_t`
Channels number in a conversion group.
- `typedef struct ADCDriver ADCDriver`
Type of a structure representing an ADC driver.
- `typedef void(* adccallback_t)(ADCDriver *adcp, adcsample_t *buffer, size_t n)`
ADC notification callback type.
- `typedef void(* adcerrorcallback_t)(ADCDriver *adcp, adcerror_t err)`
ADC error callback type.

Enumerations

- `enum adcerror_t { ADC_ERR_DMAFAILURE = 0, ADC_ERR_OVERFLOW = 1 }`
Possible ADC failure causes.

13.5 can.c File Reference

13.5.1 Detailed Description

CAN Driver code.

```
#include "ch.h"
#include "hal.h"
```

Functions

- void **canInit** (void)
CAN Driver initialization.
- void **canObjectInit** (**CANDriver** *canp)
*Initializes the standard part of a **CANDriver** structure.*
- void **canStart** (**CANDriver** *canp, const **CANConfig** *config)
Configures and activates the CAN peripheral.
- void **canStop** (**CANDriver** *canp)
Deactivates the CAN peripheral.
- **msg_t canTransmit** (**CANDriver** *canp, **canmbx_t** mailbox, const **CANTxFrame** *ctfp, **systime_t** timeout)
Can frame transmission.
- **msg_t canReceive** (**CANDriver** *canp, **canmbx_t** mailbox, **CANRxFrame** *crfp, **systime_t** timeout)
Can frame receive.
- void **canSleep** (**CANDriver** *canp)
Enters the sleep mode.
- void **canWakeup** (**CANDriver** *canp)
Enforces leaving the sleep mode.

13.6 can.h File Reference

13.6.1 Detailed Description

CAN Driver macros and structures. #include "can_lld.h"

Functions

- void **canInit** (void)
CAN Driver initialization.
- void **canObjectInit** (**CANDriver** *canp)
*Initializes the standard part of a **CANDriver** structure.*
- void **canStart** (**CANDriver** *canp, const **CANConfig** *config)
Configures and activates the CAN peripheral.
- void **canStop** (**CANDriver** *canp)
Deactivates the CAN peripheral.
- **msg_t canTransmit** (**CANDriver** *canp, **canmbx_t** mailbox, const **CANTxFrame** *ctfp, **systime_t** timeout)
Can frame transmission.
- **msg_t canReceive** (**CANDriver** *canp, **canmbx_t** mailbox, **CANRxFrame** *crfp, **systime_t** timeout)
Can frame receive.

Defines

- #define **CAN_ANY_MAILBOX** 0
Special mailbox identifier.

CAN status flags

- #define CAN_LIMIT_WARNING 1
Errors rate warning.
- #define CAN_LIMIT_ERROR 2
Errors rate error.
- #define CAN_BUS_OFF_ERROR 4
Bus off condition reached.
- #define CAN_FRAMING_ERROR 8
Framing error of some kind on the CAN bus.
- #define CAN_OVERFLOW_ERROR 16
Overflow in receive queue.

CAN configuration options

- #define CAN_USE_SLEEP_MODE TRUE
Sleep mode related APIs inclusion switch.

Macro Functions

- #define CAN_MAILBOX_TO_MASK(mbx) (1 << ((mbx) - 1))
Converts a mailbox index to a bit mask.

Enumerations

- enum canstate_t {
CAN_UNINIT = 0, CAN_STOP = 1, CAN_STARTING = 2, CAN_READY = 3,
CAN_SLEEP = 4 }
Driver state machine possible states.

13.7 can_lld.c File Reference

13.7.1 Detailed Description

CAN Driver subsystem low level driver source template.

```
#include "ch.h"
#include "hal.h"
```

Functions

- void can_lld_init (void)
Low level CAN driver initialization.
- void can_lld_start (CANDriver *canp)
Configures and activates the CAN peripheral.
- void can_lld_stop (CANDriver *canp)
Deactivates the CAN peripheral.
- bool_t can_lld_is_tx_empty (CANDriver *canp, canmbx_t mailbox)
Determines whether a frame can be transmitted.
- void can_lld_transmit (CANDriver *canp, canmbx_t mailbox, const CANTxFFrame *ctfp)
Inserts a frame into the transmit queue.
- bool_t can_lld_is_rx_nonempty (CANDriver *canp, canmbx_t mailbox)
Determines whether a frame has been received.
- void can_lld_receive (CANDriver *canp, canmbx_t mailbox, CANRxFrame *crfp)

- void **can_lld_sleep** (CANDriver *canp)

Enters the sleep mode.
- void **can_lld_wakeup** (CANDriver *canp)

Enforces leaving the sleep mode.

Variables

- CANDriver **CAND1**

CAN1 driver identifier.

13.8 can_lld.h File Reference

13.8.1 Detailed Description

CAN Driver subsystem low level driver header template.

Data Structures

- struct **CANTxFrame**

CAN transmission frame.
- struct **CANRxFrame**

CAN received frame.
- struct **CANFilter**

CAN filter.
- struct **CANConfig**

Driver configuration structure.
- struct **CANDriver**

Structure representing an CAN driver.

Functions

- void **can_lld_init** (void)

Low level CAN driver initialization.
- void **can_lld_start** (CANDriver *canp)

Configures and activates the CAN peripheral.
- void **can_lld_stop** (CANDriver *canp)

Deactivates the CAN peripheral.
- **bool_t** **can_lld_is_tx_empty** (CANDriver *canp, **canmbx_t** mailbox)

Determines whether a frame can be transmitted.
- void **can_lld_transmit** (CANDriver *canp, **canmbx_t** mailbox, const **CANTxFrame** *ctfp)

Inserts a frame into the transmit queue.
- **bool_t** **can_lld_is_rx_nonempty** (CANDriver *canp, **canmbx_t** mailbox)

Determines whether a frame has been received.
- void **can_lld_receive** (CANDriver *canp, **canmbx_t** mailbox, **CANRxFrame** *crfp)

Receives a frame from the input queue.

Defines

- `#define CAN_SUPPORTS_SLEEP TRUE`
This switch defines whether the driver implementation supports a low power switch mode with automatic an wakeup feature.
- `#define CAN_TX_MAILBOXES 3`
This implementation supports three transmit mailboxes.
- `#define CAN_RX_MAILBOXES 2`
This implementation supports two receive mailboxes.

Configuration options

- `#define PLATFORM_CAN_USE_CAN1 FALSE`
CAN1 driver enable switch.

Typedefs

- `typedef uint32_t canmbx_t`
Type of a transmission mailbox index.

13.9 ch.cpp File Reference

13.9.1 Detailed Description

C++ wrapper code. `#include "ch.hpp"`

Namespaces

- namespace `chibios_rt`
ChibiOS kernel-related classes and interfaces.

13.10 ch.h File Reference

13.10.1 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. `#include "chconf.h"`

```
#include "ctypes.h"
#include "chlists.h"
#include "chcore.h"
#include "chsys.h"
#include "chvt.h"
#include "chsched.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 "chthreads.h"
#include "chdynamic.h"
#include "chregistry.h"
#include "chinline.h"
#include "chqueues.h"
#include "chstreams.h"
#include "chfiles.h"
#include "chdebug.h"
```

Functions

- void **_idle_thread** (void *p)
This function implements the idle thread infinite loop.

Defines

- #define **_CHIBIOS_RT_**
ChibiOS/RT identification macro.
- #define **CH_KERNEL_VERSION** "2.6.9"
Kernel version string.

Kernel version

- #define **CH_KERNEL_MAJOR** 2
Kernel version major number.
- #define **CH_KERNEL_MINOR** 6
Kernel version minor number.
- #define **CH_KERNEL_PATCH** 9
Kernel version patch number.

Common constants

- #define **FALSE** 0
Generic 'false' boolean constant.
- #define **TRUE** (!FALSE)
Generic 'true' boolean constant.
- #define **CH_SUCCESS** FALSE
Generic success constant.
- #define **CH_FAILED** TRUE
Generic failure constant.

13.11 ch.hpp File Reference

13.11.1 Detailed Description

C++ wrapper classes and definitions. #include <ch.h>

Data Structures

- class [chibios_rt::System](#)
Class encapsulating the base system functionalities.
- class [chibios_rt::Core](#)
Class encapsulating the base system functionalities.
- class [chibios_rt::Timer](#)
Timer class.
- class [chibios_rt::ThreadReference](#)
Thread reference class.
- class [chibios_rt::BaseThread](#)
Abstract base class for a ChibiOS/RT thread.
- class [chibios_rt::BaseStaticThread< N >](#)
Static threads template class.
- class [chibios_rt::CounterSemaphore](#)
Class encapsulating a semaphore.
- class [chibios_rt::BinarySemaphore](#)
Class encapsulating a binary semaphore.
- class [chibios_rt::Mutex](#)
Class encapsulating a mutex.
- class [chibios_rt::CondVar](#)
Class encapsulating a conditional variable.
- class [chibios_rt::EvtListener](#)
Class encapsulating an event listener.
- class [chibios_rt::EvtSource](#)
Class encapsulating an event source.
- class [chibios_rt::InQueue](#)
Class encapsulating an input queue.
- class [chibios_rt::InQueueBuffer< N >](#)
Template class encapsulating an input queue and its buffer.
- class [chibios_rt::OutQueue](#)
Class encapsulating an output queue.
- class [chibios_rt::OutQueueBuffer< N >](#)
Template class encapsulating an output queue and its buffer.
- class [chibios_rt::Mailbox](#)
Class encapsulating a mailbox.
- class [chibios_rt::MailboxBuffer< N >](#)
Template class encapsulating a mailbox and its messages buffer.
- class [chibios_rt::MemoryPool](#)
Class encapsulating a mailbox.
- class [chibios_rt::ObjectsPool< T, N >](#)
Template class encapsulating a memory pool and its elements.
- class [chibios_rt::BaseSequentialStreamInterface](#)
Interface of a [BaseSequentialStream](#).

Namespaces

- namespace `chibios_rt`
ChibiOS kernel-related classes and interfaces.

13.12 chbsem.h File Reference

13.12.1 Detailed Description

Binary semaphores structures and macros.

Data Structures

- struct `BinarySemaphore`
Binary semaphore type.

Defines

- `#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) BinarySemaphore name = _BSEMAPHORE_DATA(name, taken)`
Static semaphore initializer.

Macro Functions

- `#define chBSemInit(bsp, taken) chSemInit(&(bsp)->bs_sem, (taken) ? 0 : 1)`
Initializes a binary semaphore.
- `#define chBSemWait(bsp) chSemWait(&(bsp)->bs_sem)`
Wait operation on the binary semaphore.
- `#define chBSemWaitS(bsp) chSemWaitS(&(bsp)->bs_sem)`
Wait operation on the binary semaphore.
- `#define chBSemWaitTimeout(bsp, time) chSemWaitTimeout(&(bsp)->bs_sem, (time))`
Wait operation on the binary semaphore.
- `#define chBSemWaitTimeoutS(bsp, time) chSemWaitTimeoutS(&(bsp)->bs_sem, (time))`
Wait operation on the binary semaphore.
- `#define chBSemReset(bsp, taken) chSemReset(&(bsp)->bs_sem, (taken) ? 0 : 1)`
Reset operation on the binary semaphore.
- `#define chBSemResetI(bsp, taken) chSemResetI(&(bsp)->bs_sem, (taken) ? 0 : 1)`
Reset operation on the binary semaphore.
- `#define chBSemSignal(bsp)`
Performs a signal operation on a binary semaphore.
- `#define chBSemSignall(bsp)`
Performs a signal operation on a binary semaphore.
- `#define chBSemGetStateI(bsp) ((bsp)->bs_sem.s_cnt > 0 ? FALSE : TRUE)`
Returns the binary semaphore current state.

13.13 chcond.c File Reference

13.13.1 Detailed Description

Condition Variables code. `#include "ch.h"`

Functions

- void **chCondInit** (**CondVar** *cp)
*Initializes a **CondVar** structure.*
- void **chCondSignal** (**CondVar** *cp)
Signals one thread that is waiting on the condition variable.
- void **chCondSignall** (**CondVar** *cp)
Signals one thread that is waiting on the condition variable.
- void **chCondBroadcast** (**CondVar** *cp)
Signals all threads that are waiting on the condition variable.
- void **chCondBroadcastl** (**CondVar** *cp)
Signals all threads that are waiting on the condition variable.
- **msg_t chCondWait** (**CondVar** *cp)
Waits on the condition variable releasing the mutex lock.
- **msg_t chCondWaitS** (**CondVar** *cp)
Waits on the condition variable releasing the mutex lock.
- **msg_t chCondWaitTimeout** (**CondVar** *cp, **systime_t** time)
Waits on the condition variable releasing the mutex lock.
- **msg_t chCondWaitTimeoutS** (**CondVar** *cp, **systime_t** time)
Waits on the condition variable releasing the mutex lock.

13.14 chcond.h File Reference

13.14.1 Detailed Description

Condition Variables macros and structures.

Data Structures

- struct **CondVar**
CondVar structure.

Functions

- void **chCondInit** (**CondVar** *cp)
*Initializes a **CondVar** structure.*
- void **chCondSignal** (**CondVar** *cp)
Signals one thread that is waiting on the condition variable.
- void **chCondSignall** (**CondVar** *cp)
Signals one thread that is waiting on the condition variable.
- void **chCondBroadcast** (**CondVar** *cp)
Signals all threads that are waiting on the condition variable.
- void **chCondBroadcastl** (**CondVar** *cp)
Signals all threads that are waiting on the condition variable.
- **msg_t chCondWait** (**CondVar** *cp)
Waits on the condition variable releasing the mutex lock.
- **msg_t chCondWaitS** (**CondVar** *cp)
Waits on the condition variable releasing the mutex lock.
- **msg_t chCondWaitTimeout** (**CondVar** *cp, **systime_t** time)
Waits on the condition variable releasing the mutex lock.
- **msg_t chCondWaitTimeoutS** (**CondVar** *cp, **systime_t** time)
Waits on the condition variable releasing the mutex lock.

Defines

- `#define _CONDVAR_DATA(name) {_THREADSQUEUE_DATA(name.c_queue)}`
Data part of a static condition variable initializer.
- `#define CONDVAR_DECL(name) CondVar name = _CONDVAR_DATA(name)`
Static condition variable initializer.

Typedefs

- `typedef struct CondVar CondVar`
CondVar structure.

13.15 chconf.h File Reference

13.15.1 Detailed Description

Configuration file template. A copy of this file must be placed in each project directory, it contains the application specific kernel settings.

Defines

Kernel parameters and options

- `#define CH_FREQUENCY 1000`
System tick frequency.
- `#define CH_TIME_QUANTUM 20`
Round robin interval.
- `#define CH_MEMCORE_SIZE 0`
Managed RAM size.
- `#define CH_NO_IDLE_THREAD FALSE`
Idle thread automatic spawn suppression.

Performance options

- `#define CH_OPTIMIZE_SPEED TRUE`
OS optimization.

Subsystem options

- `#define CH_USE_REGISTRY TRUE`
Threads registry APIs.
- `#define CH_USE_WAITEXIT TRUE`
Threads synchronization APIs.
- `#define CH_USE_SEMAPHORES TRUE`
Semaphores APIs.
- `#define CH_USE_SEMAPHORES_PRIORITY FALSE`
Semaphores queuing mode.
- `#define CH_USE_SEMSW TRUE`
Atomic semaphore API.
- `#define CH_USE_MUTEXES TRUE`
Mutexes APIs.
- `#define CH_USE_CONDVAR TRUE`
Conditional Variables APIs.
- `#define CH_USE_CONDVAR_TIMEOUT TRUE`
Conditional Variables APIs with timeout.

- #define `CH_USE_EVENTS` TRUE
Events Flags APIs.
- #define `CH_USE_EVENTS_TIMEOUT` TRUE
Events Flags APIs with timeout.
- #define `CH_USE_MESSAGES` TRUE
Synchronous Messages APIs.
- #define `CH_USE_MESSAGES_PRIORITY` FALSE
Synchronous Messages queuing mode.
- #define `CH_USE_MAILBOXES` TRUE
Mailboxes APIs.
- #define `CH_USE_QUEUES` TRUE
I/O Queues APIs.
- #define `CH_USE_MEMCORE` TRUE
Core Memory Manager APIs.
- #define `CH_USE_HEAP` TRUE
Heap Allocator APIs.
- #define `CH_USE_MALLOC_HEAP` FALSE
C-runtime allocator.
- #define `CH_USE_MEMPOOLS` TRUE
Memory Pools Allocator APIs.
- #define `CH_USE_DYNAMIC` TRUE
Dynamic Threads APIs.

Debug options

- #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` TRUE
Debug option, threads profiling.

Kernel hooks

- #define `THREAD_EXT_FIELDS`
Threads descriptor structure extension.
- #define `THREAD_EXT_INIT_HOOK(tp)`
Threads initialization hook.
- #define `THREAD_EXT_EXIT_HOOK(tp)`
Threads finalization hook.
- #define `THREAD_CONTEXT_SWITCH_HOOK(ntp, otp)`
Context switch hook.
- #define `IDLE_LOOP_HOOK()`
Idle Loop hook.
- #define `SYSTEM_TICK_EVENT_HOOK()`
System tick event hook.
- #define `SYSTEM_HALT_HOOK()`
System halt hook.

13.16 chcore.c File Reference

13.16.1 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. `#include "ch.h"`

Functions

- void `port_init` (void)
Port-related initialization code.
- void `port_lock` (void)
Kernel-lock action.
- void `port_unlock` (void)
Kernel-unlock action.
- void `port_lock_from_isr` (void)
Kernel-lock action from an interrupt handler.
- void `port_unlock_from_isr` (void)
Kernel-unlock action from an interrupt handler.
- void `port_disable` (void)
Disables all the interrupt sources.
- void `port_suspend` (void)
Disables the interrupt sources below kernel-level priority.
- void `port_enable` (void)
Enables all the interrupt sources.
- void `port_wait_for_interrupt` (void)
Enters an architecture-dependent IRQ-waiting mode.
- void `port_halt` (void)
Halts the system.
- void `port_switch` (`Thread` *ntp, `Thread` *otp)
Performs a context switch between two threads.

13.17 chcore.h File Reference

13.17.1 Detailed Description

Port related template macros and structures. This file is a template of the system driver macros provided by a port.

Data Structures

- struct `extctx`
Interrupt saved context.
- struct `intctx`
System saved context.
- struct `context`
Platform dependent part of the `Thread` structure.

Functions

- void `port_init` (void)
Port-related initialization code.
- void `port_lock` (void)
Kernel-lock action.
- void `port_unlock` (void)
Kernel-unlock action.
- void `port_lock_from_isr` (void)
Kernel-lock action from an interrupt handler.
- void `port_unlock_from_isr` (void)
Kernel-unlock action from an interrupt handler.
- void `port_disable` (void)
Disables all the interrupt sources.
- void `port_suspend` (void)
Disables the interrupt sources below kernel-level priority.
- void `port_enable` (void)
Enables all the interrupt sources.
- void `port_wait_for_interrupt` (void)
Enters an architecture-dependent IRQ-waiting mode.
- void `port_halt` (void)
Halts the system.
- void `port_switch` (`Thread` *ntp, `Thread` *otp)
Performs a context switch between two threads.

Defines

- `#define PORT_IDLE_THREAD_STACK_SIZE 0`
Stack size for the system idle thread.
- `#define PORT_INT_REQUIRED_STACK 0`
Per-thread stack overhead for interrupts servicing.
- `#define CH_ARCHITECTURE_XXX`
Unique macro for the implemented architecture.
- `#define CH_ARCHITECTURE_NAME ""`
Name of the implemented architecture.
- `#define CH_ARCHITECTURE_VARIANT_NAME ""`
Name of the architecture variant (optional).
- `#define CH_COMPILER_NAME "GCC"`
Name of the compiler supported by this port.
- `#define CH_PORT_INFO ""`
Port-specific information string.
- `#define SETUP_CONTEXT(workspace, wsize, pf, arg)`
Platform dependent part of the `chThdCreateI()` API.
- `#define STACK_ALIGN(n) (((n) - 1) | (sizeof(stkalign_t) - 1)) + 1`
Enforces a correct alignment for a stack area size value.
- `#define THD_WA_SIZE(n)`
Computes the thread working area global size.
- `#define WORKING_AREA(s, n) stkalign_t s[THD_WA_SIZE(n) / sizeof(stkalign_t)]`
Static working area allocation.
- `#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.

Typedefs

- `typedef uint8_t stkalign_t`
Base type for stack and memory alignment.

13.18 chdebug.c File Reference

13.18.1 Detailed Description

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 _trace_init (void)`
Trace circular buffer subsystem initialization.
- `void dbg_trace (Thread *otp)`
Inserts in the circular debug trace buffer a context switch record.
- `void chDbgPanic (const char *msg)`
Prints a panic message on the console and then halts the system.

Variables

- `cnt_t dbg_isr_cnt`
ISR nesting level.
- `cnt_t dbg_lock_cnt`
Lock nesting level.
- `ch_trace_buffer_t dbg_trace_buffer`
Public trace buffer.
- `const char * dbg_panic_msg`
Pointer to the panic message.

13.19 chdebug.h File Reference

13.19.1 Detailed Description

Debug macros and structures.

Data Structures

- struct `ch_swc_event_t`
Trace buffer record.
- struct `ch_trace_buffer_t`
Trace buffer header.

Functions

- `void _trace_init (void)`
Trace circular buffer subsystem initialization.
- `void dbg_trace (Thread *otp)`
Inserts in the circular debug trace buffer a context switch record.

Defines

Debug related settings

- `#define CH_TRACE_BUFFER_SIZE 64`
Trace buffer entries.
- `#define CH_STACK_FILL_VALUE 0x55`
Fill value for thread stack area in debug mode.
- `#define CH_THREAD_FILL_VALUE 0xFF`
Fill value for thread area in debug mode.

Macro Functions

- `#define chDbgCheck(c, func)`
Function parameters check.
- `#define chDbgAssert(c, m, r)`
Condition assertion.

13.20 chdynamic.c File Reference

13.20.1 Detailed Description

Dynamic threads code. #include "ch.h"

Functions

- `Thread * chThdAddRef (Thread *tp)`
Adds a reference to a thread object.
- `void chThdRelease (Thread *tp)`
Releases a reference to a thread object.
- `Thread * chThdCreateFromHeap (MemoryHeap *heapp, size_t size, tprio_t prio, tfunc_t pf, void *arg)`
Creates a new thread allocating the memory from the heap.
- `Thread * chThdCreateFromMemoryPool (MemoryPool *mp, tprio_t prio, tfunc_t pf, void *arg)`
Creates a new thread allocating the memory from the specified memory pool.

13.21 chdynamic.h File Reference

13.21.1 Detailed Description

Dynamic threads macros and structures.

Functions

- `Thread * chThdAddRef (Thread *tp)`
Adds a reference to a thread object.
- `void chThdRelease (Thread *tp)`
Releases a reference to a thread object.
- `Thread * chThdCreateFromHeap (MemoryHeap *heapp, size_t size, tprio_t prio, tfunc_t pf, void *arg)`
Creates a new thread allocating the memory from the heap.
- `Thread * chThdCreateFromMemoryPool (MemoryPool *mp, tprio_t prio, tfunc_t pf, void *arg)`
Creates a new thread allocating the memory from the specified memory pool.

13.22 chevents.c File Reference

13.22.1 Detailed Description

Events code. #include "ch.h"

Functions

- `void chEvtRegisterMask (EventSource *esp, EventListener *elp, eventmask_t mask)`
Registers an Event Listener on an Event Source.
- `void chEvtUnregister (EventSource *esp, EventListener *elp)`
Unregisters an Event Listener from its Event Source.
- `eventmask_t chEvtGetAndClearEvents (eventmask_t mask)`
Clears the pending events specified in the mask.

- `eventmask_t chEvtAddEvents (eventmask_t mask)`
*Adds (OR) a set of event flags on the current thread, this is **much** faster than using `chEvtBroadcast()` or `chEvtSignal()`.*
- `void chEvtBroadcastFlagsI (EventSource *esp, flagsmask_t flags)`
Signals all the Event Listeners registered on the specified Event Source.
- `flagsmask_t chEvtGetAndClearFlags (EventListener *elp)`
Returns the flags associated to an `EventListener`.
- `void chEvtSignal (Thread *tp, eventmask_t mask)`
Adds a set of event flags directly to specified `Thread`.
- `void chEvtSignall (Thread *tp, eventmask_t mask)`
Adds a set of event flags directly to specified `Thread`.
- `void chEvtBroadcastFlags (EventSource *esp, flagsmask_t flags)`
Signals all the Event Listeners registered on the specified Event Source.
- `flagsmask_t chEvtGetAndClearFlagsI (EventListener *elp)`
Returns the flags associated to an `EventListener`.
- `void chEvtDispatch (const evhandler_t *handlers, eventmask_t mask)`
Invokes the event handlers associated to an event flags mask.
- `eventmask_t chEvtWaitOne (eventmask_t mask)`
Waits for exactly one of the specified events.
- `eventmask_t chEvtWaitAny (eventmask_t mask)`
Waits for any of the specified events.
- `eventmask_t chEvtWaitAll (eventmask_t mask)`
Waits for all the specified events.
- `eventmask_t chEvtWaitOneTimeout (eventmask_t mask, systime_t time)`
Waits for exactly one of the specified events.
- `eventmask_t chEvtWaitAnyTimeout (eventmask_t mask, systime_t time)`
Waits for any of the specified events.
- `eventmask_t chEvtWaitAllTimeout (eventmask_t mask, systime_t time)`
Waits for all the specified events.

13.23 chevents.h File Reference

13.23.1 Detailed Description

Events macros and structures.

Data Structures

- struct `EventListener`
Event Listener structure.
- struct `EventSource`
Event Source structure.

Functions

- `void chEvtRegisterMask (EventSource *esp, EventListener *elp, eventmask_t mask)`
Registers an Event Listener on an Event Source.
- `void chEvtUnregister (EventSource *esp, EventListener *elp)`
Unregisters an Event Listener from its Event Source.
- `eventmask_t chEvtGetAndClearEvents (eventmask_t mask)`

- Clears the pending events specified in the mask.
- `eventmask_t chEvtAddEvents (eventmask_t mask)`

Adds (OR) a set of event flags on the current thread, this is ***much*** faster than using `chEvtBroadcast ()` or `chEvtSignal ()`.
- `flagsmask_t chEvtGetAndClearFlags (EventListener *elp)`

Returns the flags associated to an `EventListener`.
- `flagsmask_t chEvtGetAndClearFlagsI (EventListener *elp)`

Returns the flags associated to an `EventListener`.
- `void chEvtSignal (Thread *tp, eventmask_t mask)`

Adds a set of event flags directly to specified `Thread`.
- `void chEvtSignalI (Thread *tp, eventmask_t mask)`

Adds a set of event flags directly to specified `Thread`.
- `void chEvtBroadcastFlags (EventSource *esp, flagsmask_t flags)`

Signals all the Event Listeners registered on the specified Event Source.
- `void chEvtBroadcastFlagsI (EventSource *esp, flagsmask_t flags)`

Signals all the Event Listeners registered on the specified Event Source.
- `void chEvtDispatch (const evhandler_t *handlers, eventmask_t mask)`

Invokes the event handlers associated to an event flags mask.
- `eventmask_t chEvtWaitOneTimeout (eventmask_t mask, systime_t time)`

Waits for exactly one of the specified events.
- `eventmask_t chEvtWaitAnyTimeout (eventmask_t mask, systime_t time)`

Waits for any of the specified events.
- `eventmask_t chEvtWaitAllTimeout (eventmask_t mask, systime_t time)`

Waits for all the specified events.

Defines

- `#define _EVENTSOURCE_DATA(name) { (void *)(&name)}`

Data part of a static event source initializer.
- `#define EVENTSOURCE DECL(name) EventSource name = _EVENTSOURCE_DATA(name)`

Static event source initializer.
- `#define ALL_EVENTS ((eventmask_t)-1)`

All events allowed mask.
- `#define EVENT_MASK(eid) ((eventmask_t)(1 << (eid)))`

Returns an event mask from an event identifier.

Macro Functions

- `#define chEvtRegister(esp, elp, eid) chEvtRegisterMask(esp, elp, EVENT_MASK(eid))`

Registers an Event Listener on an Event Source.
- `#define chEvtInit(esp) ((esp)->es_next = (EventListener *)(void *)(esp))`

Initializes an Event Source.
- `#define chEvtIsListeningI(esp) ((void *)(esp) != (void *)(esp)->es_next)`

Verifies if there is at least one `EventListener` registered.
- `#define chEvtBroadcast(esp) chEvtBroadcastFlags(esp, 0)`

Signals all the Event Listeners registered on the specified Event Source.
- `#define chEvtBroadcastI(esp) chEvtBroadcastFlagsI(esp, 0)`

Signals all the Event Listeners registered on the specified Event Source.

Typedefs

- `typedef struct EventSource EventSource`

Event Source structure.
- `typedef void(* evhandler_t)(eventid_t)`

Event Handler callback function.

13.24 chfiles.h File Reference

13.24.1 Detailed Description

Data files. This header defines abstract interfaces useful to access generic data files in a standardized way.

Data Structures

- struct [BaseFileStreamVMT](#)
BaseFileStream virtual methods table.
- struct [BaseFileStream](#)
Base file stream class.

Defines

- #define [FILE_OK](#) 0
No error return code.
- #define [FILE_ERROR](#) 0xFFFFFFFFFUL
Error code from the file stream methods.
- #define [_base_file_stream_methods](#)
BaseFileStream specific methods.
- #define [_base_file_stream_data_base_sequential_stream_data](#)
BaseFileStream specific data.

Macro Functions (BaseFileStream)

- #define [chFileStreamClose\(ip\)](#) ((ip)->vmt->close(ip))
Base file Stream close.
- #define [chFileStreamGetError\(ip\)](#) ((ip)->vmt->geterror(ip))
Returns an implementation dependent error code.
- #define [chFileStreamGetSize\(ip\)](#) ((ip)->vmt->getsize(ip))
Returns the current file size.
- #define [chFileStreamGetPosition\(ip\)](#) ((ip)->vmt->getposition(ip))
Returns the current file pointer position.
- #define [chFileStreamSeek\(ip, offset\)](#) ((ip)->vmt->lseek(ip, offset))
Moves the file current pointer to an absolute position.

Typedefs

- typedef uint32_t [fileoffset_t](#)
File offset type.

13.25 chheap.c File Reference

13.25.1 Detailed Description

Hooks code. #include "ch.h"

Functions

- void `_heap_init` (void)
Initializes the default heap.
- void `chHeapInit` (`MemoryHeap` *heapp, void *buf, size_t size)
Initializes a memory heap from a static memory area.
- void * `chHeapAlloc` (`MemoryHeap` *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` (`MemoryHeap` *heapp, size_t *sizep)
Reports the heap status.

13.26 chheap.h File Reference

13.26.1 Detailed Description

Hooks macros and structures.

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 `chHeapInit` (`MemoryHeap` *heapp, void *buf, size_t size)
Initializes a memory heap from a static memory area.
- void * `chHeapAlloc` (`MemoryHeap` *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` (`MemoryHeap` *heapp, size_t *sizep)
Reports the heap status.

13.27 chinline.h File Reference

13.27.1 Detailed Description

Kernel inlined functions. In this file there are a set of inlined functions if the `CH_OPTIMIZE_SPEED` is enabled.

13.28 chlists.c File Reference

13.28.1 Detailed Description

Thread queues/lists code. `#include "ch.h"`

Functions

- `void prio_insert (Thread *tp, ThreadsQueue *tqp)`
Inserts a thread into a priority ordered queue.
- `void queue_insert (Thread *tp, ThreadsQueue *tqp)`
Inserts a Thread into a queue.
- `Thread * fifo_remove (ThreadsQueue *tqp)`
Removes the first-out Thread from a queue and returns it.
- `Thread * lifo_remove (ThreadsQueue *tqp)`
Removes the last-out Thread from a queue and returns it.
- `Thread * dequeue (Thread *tp)`
Removes a Thread from a queue and returns it.
- `void list_insert (Thread *tp, ThreadsList *tlp)`
Pushes a Thread on top of a stack list.
- `Thread * list_remove (ThreadsList *tlp)`
Pops a Thread from the top of a stack list and returns it.

13.29 chlists.h File Reference

13.29.1 Detailed Description

`Thread` queues/lists macros and structures.

Note

All the macros present in this module, while public, are not an OS API and should not be directly used in the user applications code.

Data Structures

- `struct ThreadsQueue`
Generic threads bidirectional linked list header and element.
- `struct ThreadsList`
Generic threads single link list, it works like a stack.

Functions

- `void prio_insert (Thread *tp, ThreadsQueue *tqp)`
Inserts a thread into a priority ordered queue.
- `void queue_insert (Thread *tp, ThreadsQueue *tqp)`
Inserts a Thread into a queue.
- `Thread * fifo_remove (ThreadsQueue *tqp)`
Removes the first-out Thread from a queue and returns it.
- `Thread * lifo_remove (ThreadsQueue *tqp)`
Removes the last-out Thread from a queue and returns it.
- `Thread * dequeue (Thread *tp)`
Removes a Thread from a queue and returns it.
- `void list_insert (Thread *tp, ThreadsList *tlp)`
Pushes a Thread on top of a stack list.
- `Thread * list_remove (ThreadsList *tlp)`
Pops a Thread from the top of a stack list and returns it.

Defines

- `#define queue_init(tqp) ((tqp)->p_next = (tqp)->p_prev = (Thread *)(tqp));`
Threads queue initialization.
- `#define list_init(tlp) ((tlp)->p_next = (Thread *)(tlp))`
Threads list initialization.
- `#define isempty(p) ((p)->p_next == (Thread *)(p))`
Evaluates to TRUE if the specified threads queue or list is empty.
- `#define notempty(p) ((p)->p_next != (Thread *)(p))`
Evaluates to TRUE if the specified threads queue or list is not empty.
- `#define _THREADSQUEUE_DATA(name) { (Thread *)&name, (Thread *)&name }`
Data part of a static threads queue initializer.
- `#define THREADSQUEUE_DECL(name) ThreadsQueue name = _THREADSQUEUE_DATA(name)`
Static threads queue initializer.

13.30 chmboxes.c File Reference

13.30.1 Detailed Description

Mailboxes code. `#include "ch.h"`

Functions

- `void chMBInit (Mailbox *mbp, msg_t *buf, cnt_t n)`
Initializes a Mailbox object.
- `void chMBReset (Mailbox *mbp)`
Resets a Mailbox object.
- `msg_t chMBPost (Mailbox *mbp, msg_t msg, systime_t time)`
Posts a message into a mailbox.
- `msg_t chMBPostS (Mailbox *mbp, msg_t msg, systime_t time)`
Posts a message into a mailbox.
- `msg_t chMBPostI (Mailbox *mbp, msg_t msg)`
Posts a message into a mailbox.
- `msg_t chMBPostAhead (Mailbox *mbp, msg_t msg, systime_t time)`
Posts an high priority message into a mailbox.
- `msg_t chMBPostAheadS (Mailbox *mbp, msg_t msg, systime_t time)`
Posts an high priority message into a mailbox.
- `msg_t chMBPostAheadI (Mailbox *mbp, msg_t msg)`
Posts an high priority message into a mailbox.
- `msg_t chMBFetch (Mailbox *mbp, msg_t *msgp, systime_t time)`
Retrieves a message from a mailbox.
- `msg_t chMBFetchS (Mailbox *mbp, msg_t *msgp, systime_t time)`
Retrieves a message from a mailbox.
- `msg_t chMBFetchI (Mailbox *mbp, msg_t *msgp)`
Retrieves a message from a mailbox.

13.31 chmboxes.h File Reference

13.31.1 Detailed Description

Mailboxes macros and structures.

Data Structures

- struct `Mailbox`

Structure representing a mailbox object.

Functions

- void `chMBInit` (`Mailbox *mbp, msg_t *buf, cnt_t n`)
Initializes a `Mailbox` object.
- void `chMBReset` (`Mailbox *mbp`)
Resets a `Mailbox` object.
- `msg_t chMBPost` (`Mailbox *mbp, msg_t msg, systime_t time`)
Posts a message into a mailbox.
- `msg_t chMBPostS` (`Mailbox *mbp, msg_t msg, systime_t time`)
Posts a message into a mailbox.
- `msg_t chMBPostI` (`Mailbox *mbp, msg_t msg`)
Posts a message into a mailbox.
- `msg_t chMBPostAhead` (`Mailbox *mbp, msg_t msg, systime_t time`)
Posts an high priority message into a mailbox.
- `msg_t chMBPostAheadS` (`Mailbox *mbp, msg_t msg, systime_t time`)
Posts an high priority message into a mailbox.
- `msg_t chMBPostAheadI` (`Mailbox *mbp, msg_t msg`)
Posts an high priority message into a mailbox.
- `msg_t chMBFetch` (`Mailbox *mbp, msg_t *msgp, systime_t time`)
Retrieves a message from a mailbox.
- `msg_t chMBFetchS` (`Mailbox *mbp, msg_t *msgp, systime_t time`)
Retrieves a message from a mailbox.
- `msg_t chMBFetchI` (`Mailbox *mbp, msg_t *msgp`)
Retrieves a message from a mailbox.

Defines

- `#define _MAILBOX_DATA(name, buffer, size)`
Data part of a static mailbox initializer.
- `#define MAILBOX_DECL(name, buffer, size) Mailbox name = _MAILBOX_DATA(name, buffer, size)`
Static mailbox initializer.

Macro Functions

- `#define chMBSizel(mbp) ((mbp)->mb_top - (mbp)->mb_buffer)`
Returns the mailbox buffer size.
- `#define chMBGetFreeCountl(mbp) chSemGetCounterl(&(mbp)->mb_emptysem)`
Returns the number of free message slots into a mailbox.
- `#define chMBGetUsedCountl(mbp) chSemGetCounterl(&(mbp)->mb_fullsem)`
Returns the number of used message slots into a mailbox.
- `#define chMBPeekl(mbp) (*(mbp)->mb_rdptr)`
Returns the next message in the queue without removing it.

13.32 chmemcore.c File Reference

13.32.1 Detailed Description

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 * chCoreAlloc1 (size_t size)`
Allocates a memory block.
- `size_t chCoreStatus (void)`
Core memory status.

13.33 chmemcore.h File Reference

13.33.1 Detailed Description

Core memory manager macros and structures.

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 chCoreStatus (void)`
Core memory status.

Defines

Alignment support macros

- `#define MEM_ALIGN_SIZE sizeof(stkalign_t)`
Alignment size constant.
- `#define MEM_ALIGN_MASK (MEM_ALIGN_SIZE - 1)`
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_SIZE)`
Alignment helper macro.
- `#define MEM_IS_ALIGNED(p) (((size_t)(p) & MEM_ALIGN_MASK) == 0)`
Returns whatever a pointer or memory size is aligned to the type align_t.

Typedefs

- `typedef void *(* memgetfunc_t)(size_t size)`
Memory get function.

13.34 chmempools.c File Reference

13.34.1 Detailed Description

Memory Pools code. #include "ch.h"

Functions

- void **chPoolInit** (MemoryPool *mp, size_t size, memgetfunc_t provider)
Initializes an empty memory pool.
- void **chPoolLoadArray** (MemoryPool *mp, void *p, size_t n)
Loads a memory pool with an array of static objects.
- void * **chPoolAlloc** (MemoryPool *mp)
Allocates an object from a memory pool.
- void * **chPoolAlloc** (MemoryPool *mp)
Allocates an object from a memory pool.
- void **chPoolFree** (MemoryPool *mp, void *objp)
Releases an object into a memory pool.
- void **chPoolFree** (MemoryPool *mp, void *objp)
Releases an object into a memory pool.

13.35 chmempools.h File Reference

13.35.1 Detailed Description

Memory Pools macros and structures.

Data Structures

- struct **pool_header**
Memory pool free object header.
- struct **MemoryPool**
Memory pool descriptor.

Functions

- void **chPoolInit** (MemoryPool *mp, size_t size, memgetfunc_t provider)
Initializes an empty memory pool.
- void **chPoolLoadArray** (MemoryPool *mp, void *p, size_t n)
Loads a memory pool with an array of static objects.
- void * **chPoolAlloc** (MemoryPool *mp)
Allocates an object from a memory pool.
- void * **chPoolAlloc** (MemoryPool *mp)
Allocates an object from a memory pool.
- void **chPoolFree** (MemoryPool *mp, void *objp)
Releases an object into a memory pool.
- void **chPoolFree** (MemoryPool *mp, void *objp)
Releases an object into a memory pool.

Defines

- `#define _MEMORYPOOL_DATA(name, size, provider) {NULL, size, provider}`
Data part of a static memory pool initializer.
- `#define MEMORYPOOL_DECL(name, size, provider) MemoryPool name = _MEMORYPOOL_DATA(name, size, provider)`
Static memory pool initializer in hungry mode.

Macro Functions

- `#define chPoolAdd(mp, objp) chPoolFree(mp, objp)`
Adds an object to a memory pool.
- `#define chPoolAddl(mp, objp) chPoolFreel(mp, objp)`
Adds an object to a memory pool.

13.36 chmsg.c File Reference

13.36.1 Detailed Description

Messages code. `#include "ch.h"`

Functions

- `msg_t chMsgSend (Thread *tp, msg_t msg)`
Sends a message to the specified thread.
- `Thread * chMsgWait (void)`
Suspends the thread and waits for an incoming message.
- `void chMsgRelease (Thread *tp, msg_t msg)`
Releases a sender thread specifying a response message.

13.37 chmsg.h File Reference

13.37.1 Detailed Description

Messages macros and structures.

Functions

- `msg_t chMsgSend (Thread *tp, msg_t msg)`
Sends a message to the specified thread.
- `Thread * chMsgWait (void)`
Suspends the thread and waits for an incoming message.
- `void chMsgRelease (Thread *tp, msg_t msg)`
Releases a sender thread specifying a response message.

Defines

Macro Functions

- #define `chMsgIsPendingI`(tp) ((tp)->p_msgqueue.p_next != (Thread *)&(tp)->p_msgqueue)
Evaluates to TRUE if the thread has pending messages.
- #define `chMsgGet`(tp) ((tp)->p_msg)
Returns the message carried by the specified thread.
- #define `chMsgReleaseS`(tp, msg) chSchWakeupS(tp, msg)
Releases the thread waiting on top of the messages queue.

13.38 chmtx.c File Reference

13.38.1 Detailed Description

Mutexes code. #include "ch.h"

Functions

- void `chMtxInit` (Mutex *mp)
Initializes a Mutex structure.
- void `chMtxLock` (Mutex *mp)
Locks the specified mutex.
- void `chMtxLockS` (Mutex *mp)
Locks the specified mutex.
- bool_t `chMtxTryLock` (Mutex *mp)
Tries to lock a mutex.
- bool_t `chMtxTryLockS` (Mutex *mp)
Tries to lock a mutex.
- Mutex * `chMtxUnlock` (void)
Unlocks the next owned mutex in reverse lock order.
- Mutex * `chMtxUnlockS` (void)
Unlocks the next owned mutex in reverse lock order.
- void `chMtxUnlockAll` (void)
Unlocks all the mutexes owned by the invoking thread.

13.39 chmtx.h File Reference

13.39.1 Detailed Description

Mutexes macros and structures.

Data Structures

- struct `Mutex`
Mutex structure.

Functions

- void **chMtxInit** (**Mutex** *mp)
*Initializes a **Mutex** structure.*
- void **chMtxLock** (**Mutex** *mp)
Locks the specified mutex.
- void **chMtxLockS** (**Mutex** *mp)
Locks the specified mutex.
- **bool_t** **chMtxTryLock** (**Mutex** *mp)
Tries to lock a mutex.
- **bool_t** **chMtxTryLockS** (**Mutex** *mp)
Tries to lock a mutex.
- **Mutex** * **chMtxUnlock** (void)
Unlocks the next owned mutex in reverse lock order.
- **Mutex** * **chMtxUnlockS** (void)
Unlocks the next owned mutex in reverse lock order.
- void **chMtxUnlockAll** (void)
Unlocks all the mutexes owned by the invoking thread.

Defines

- #define **_MUTEX_DATA**(name) {_THREADSQUEUE_DATA(name.m_queue), NULL, NULL}
Data part of a static mutex initializer.
- #define **MUTEX_DECL**(name) **Mutex** name = **_MUTEX_DATA**(name)
Static mutex initializer.

Macro Functions

- #define **chMtxQueueNotEmptyS**(mp) notempty(&(mp)->m_queue)
Returns TRUE if the mutex queue contains at least a waiting thread.

TypeDefs

- typedef struct **Mutex** **Mutex**
***Mutex** structure.*

13.40 chprintf.c File Reference

13.40.1 Detailed Description

```
Mini printf-like functionality. #include "ch.h"
#include "chprintf.h"
#include "memstreams.h"
```

Functions

- void **chvprintf** (**BaseSequentialStream** *chp, const char *fmt, va_list ap)
System formatted output function.
- int **chsprintf** (char *str, size_t size, const char *fmt,...)
System formatted output function.

13.41 chprintf.h File Reference

13.41.1 Detailed Description

Mini printf-like functionality.

```
#include <stdarg.h>
```

Functions

- void **chvprintf** (**BaseSequentialStream** *chp, const char *fmt, va_list ap)
System formatted output function.
- int **chsnprintf** (char *str, size_t size, const char *fmt,...)
System formatted output function.

Defines

- #define **CHPRINTF_USE_FLOAT** FALSE
Float type support.

13.42 chqueues.c File Reference

13.42.1 Detailed Description

I/O Queues code.

```
#include "ch.h"
```

Functions

- void **chIQInit** (**InputQueue** *iqp, uint8_t *bp, size_t size, **qnotify_t** infy, void *link)
Initializes an input queue.
- void **chIQResetl** (**InputQueue** *iqp)
Resets an input queue.
- **msg_t** **chIQPutl** (**InputQueue** *iqp, uint8_t b)
Input queue write.
- **msg_t** **chIQGetTimeout** (**InputQueue** *iqp, **systime_t** time)
Input queue read with timeout.
- size_t **chIQReadTimeout** (**InputQueue** *iqp, uint8_t *bp, size_t n, **systime_t** time)
Input queue read with timeout.
- void **choQInit** (**OutputQueue** *oqp, uint8_t *bp, size_t size, **qnotify_t** onfy, void *link)
Initializes an output queue.
- void **choQResetl** (**OutputQueue** *oqp)
Resets an output queue.
- **msg_t** **chOQPutTimeout** (**OutputQueue** *oqp, uint8_t b, **systime_t** time)
Output queue write with timeout.
- **msg_t** **chOQGetl** (**OutputQueue** *oqp)
Output queue read.
- size_t **chOQWriteTimeout** (**OutputQueue** *oqp, const uint8_t *bp, size_t n, **systime_t** time)
Output queue write with timeout.

13.43 chqueues.h File Reference

13.43.1 Detailed Description

I/O Queues macros and structures.

Data Structures

- struct **GenericQueue**
Generic I/O queue structure.

Functions

- void **chIQInit** (**InputQueue** *iwp, uint8_t *bp, size_t size, **qnotify_t** infy, void *link)
Initializes an input queue.
- void **chIQResetl** (**InputQueue** *iwp)
Resets an input queue.
- **msg_t chIQPutl** (**InputQueue** *iwp, uint8_t b)
Input queue write.
- **msg_t chIQGetTimeout** (**InputQueue** *iwp, **systime_t** time)
Input queue read with timeout.
- size_t **chIQReadTimeout** (**InputQueue** *iwp, uint8_t *bp, size_t n, **systime_t** time)
Input queue read with timeout.
- void **chOQInit** (**OutputQueue** *oqp, uint8_t *bp, size_t size, **qnotify_t** onfy, void *link)
Initializes an output queue.
- void **chOQResetl** (**OutputQueue** *oqp)
Resets an output queue.
- **msg_t chOQPutTimeout** (**OutputQueue** *oqp, uint8_t b, **systime_t** time)
Output queue write with timeout.
- **msg_t chOQGetl** (**OutputQueue** *oqp)
Output queue read.
- size_t **chOQWriteTimeout** (**OutputQueue** *oqp, const uint8_t *bp, size_t n, **systime_t** time)
Output queue write with timeout.

Defines

- #define **_INPUTQUEUE_DATA**(name, buffer, size, inotify, link)
Data part of a static input queue initializer.
- #define **INPUTQUEUE_DECL**(name, buffer, size, inotify, link) **InputQueue** 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) **OutputQueue** name = **_OUTPUTQUEUE_DATA**(name, buffer, size, onotify, link)
Static output queue initializer.

Queue functions returned status value

- #define **Q_OK** RDY_OK
Operation successful.
- #define **Q_TIMEOUT** RDY_TIMEOUT
Timeout condition.
- #define **Q_RESET** RDY_RESET
Queue has been reset.
- #define **Q_EMPTY** -3
Queue empty.
- #define **Q_FULL** -4
Queue full.,

Macro Functions

- #define **chQSizeI**(qp) ((size_t)((qp)->q_top - (qp)->q_buffer))
Returns the queue's buffer size.
- #define **chQSpaceI**(qp) ((qp)->q_counter)
Queue space.
- #define **chQGetLink**(qp) ((qp)->q_link)
Returns the queue application-defined link.
- #define **chIQGetFullI**(iqp) chQSpaceI(iqp)
Returns the filled space into an input queue.
- #define **chIQGetEmptyI**(iqp) (chQSizeI(iqp) - chQSpaceI(iqp))
Returns the empty space into an input queue.
- #define **chIQIsEmptyI**(iqp) ((bool_t)(chQSpaceI(iqp) <= 0))
Evaluates to TRUE if the specified input queue is empty.
- #define **chIQIsFullI**(iqp)
Evaluates to TRUE if the specified input queue is full.
- #define **chIQGet**(iqp) chIQGetTimeout(iqp, TIME_INFINITE)
Input queue read.
- #define **chOQGetFullI**(oqp) (chQSizeI(oqp) - chQSpaceI(oqp))
Returns the filled space into an output queue.
- #define **chOQGetEmptyI**(oqp) chQSpaceI(oqp)
Returns the empty space into an output queue.
- #define **chOQIsEmptyI**(oqp)
Evaluates to TRUE if the specified output queue is empty.
- #define **chOQIsFullI**(oqp) ((bool_t)(chQSpaceI(oqp) <= 0))
Evaluates to TRUE if the specified output queue is full.
- #define **chOQPut**(oqp, b) chOQPutTimeout(oqp, b, TIME_INFINITE)
Output queue write.

Typedefs

- typedef struct **GenericQueue** GenericQueue
Type of a generic I/O queue structure.
- typedef void(* **qnotify_t**)(GenericQueue *qp)
Queue notification callback type.
- typedef **GenericQueue** InputQueue
Type of an input queue structure.
- typedef **GenericQueue** OutputQueue
Type of an output queue structure.

13.44 chregistry.c File Reference

13.44.1 Detailed Description

Threads registry code. #include "ch.h"

Functions

- `Thread * chRegFirstThread (void)`
Returns the first thread in the system.
- `Thread * chRegNextThread (Thread *tp)`
Returns the thread next to the specified one.

13.45 chregistry.h File Reference

13.45.1 Detailed Description

Threads registry macros and structures.

Data Structures

- struct `chdebug_t`
ChibiOS/RT memory signature record.

Functions

- `Thread * chRegFirstThread (void)`
Returns the first thread in the system.
- `Thread * chRegNextThread (Thread *tp)`
Returns the thread next to the specified one.

Defines

- `#define REG_REMOVE(tp)`
Removes a thread from the registry list.
- `#define REG_INSERT(tp)`
Adds a thread to the registry list.

Macro Functions

- `#define chRegSetThreadName(p) (currp->p_name = (p))`
Sets the current thread name.
- `#define chRegGetThreadName(tp) ((tp)->p_name)`
Returns the name of the specified thread.

13.46 chrtclib.c File Reference

13.46.1 Detailed Description

```
RTC time conversion utilities code. #include <time.h>
#include "ch.h"
#include "hal.h"
#include "chrtclib.h"
```

Functions

- void `rtcGetTimeTm (RTCDriver *rtcp, struct tm *timp)`
Gets raw time from RTC and converts it to canonicalized format.
- void `rtcSetTimeTm (RTCDriver *rtcp, struct tm *timp)`
Sets RTC time.
- `time_t rtcGetTimeUnixSec (RTCDriver *rtcp)`
Gets raw time from RTC and converts it to unix format.
- void `rtcSetTimeUnixSec (RTCDriver *rtcp, time_t tv_sec)`
Sets RTC time.
- `uint64_t rtcGetTimeUnixUsec (RTCDriver *rtcp)`
Gets raw time from RTC and converts it to unix format.
- `uint32_t rtcGetTimeFatFromCounter (RTCDriver *rtcp)`
Get current time in format suitable for usage in FatFS.

13.47 chrtclib.h File Reference

13.47.1 Detailed Description

RTC time conversion utilities header. #include <time.h>

Functions

- `uint32_t rtcGetTimeFat (RTCDriver *rtcp)`
Get current time in format suitable for usage in FatFS.
- void `rtcGetTimeTm (RTCDriver *rtcp, struct tm *timp)`
Gets raw time from RTC and converts it to canonicalized format.
- void `rtcSetTimeTm (RTCDriver *rtcp, struct tm *timp)`
Sets RTC time.
- `time_t rtcGetTimeUnixSec (RTCDriver *rtcp)`
Gets raw time from RTC and converts it to unix format.
- `uint64_t rtcGetTimeUnixUsec (RTCDriver *rtcp)`
Gets raw time from RTC and converts it to unix format.
- void `rtcSetTimeUnixSec (RTCDriver *rtcp, time_t tv_sec)`
Sets RTC time.

13.48 chschd.c File Reference

13.48.1 Detailed Description

Scheduler code. #include "ch.h"

Functions

- void `_scheduler_init (void)`
Scheduler initialization.
- `Thread * chSchReadyI (Thread *tp)`
Inserts a thread in the Ready List.
- void `chSchGoSleepS (tstate_t newstate)`

- **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 *ntp, msg_t msg)**

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

Performs a reschedule if a higher priority thread is runnable.
- **bool_t 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

- **ReadyList rlist**

Ready list header.

13.49 chschd.h File Reference

13.49.1 Detailed Description

Scheduler macros and structures.

Data Structures

- **struct ReadyList**

Ready list header.

Functions

- **void _scheduler_init (void)**

Scheduler initialization.
- **Thread * chSchReadyI (Thread *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 *ntp, msg_t msg)**

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

Performs a reschedule if a higher priority thread is runnable.
- **bool_t 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.

Defines

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

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

- #define **currp** rlist.r_current

Current thread pointer access macro.

- #define **setcurrp**(tp) (currp = (tp))

Current thread pointer change macro.

Wakeup status codes

- #define **RDY_OK** 0

Normal wakeup message.

- #define **RDY_TIMEOUT** -1

Wakeup caused by a timeout condition.

- #define **RDY_RESET** -2

Wakeup caused by a reset condition.

Priority constants

- #define **NOPRIO** 0

Ready list header priority.

- #define **IDLEPRIO** 1

Idle thread priority.

- #define **LOWPRIO** 2

Lowest user priority.

- #define **NORMALPRIO** 64

Normal user priority.

- #define **HIGHPRIO** 127

Highest user priority.

- #define **ABSPRIO** 255

Greatest possible priority.

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.

Macro Functions

- #define **chSchIsRescRequired()** (**firstprio**(&rlist.r_queue) > currp->p_prio)

Determines if the current thread must reschedule.

- #define **chSchCanYieldS()** (**firstprio**(&rlist.r_queue) >= currp->p_prio)

Determines if yielding is possible.

- #define **chSchDoYieldS()**

Yields the time slot.

- #define **chSchPreemption()**

Inlineable preemption code.

13.50 chsem.c File Reference

13.50.1 Detailed Description

Semaphores code. #include "ch.h"

Functions

- void **chSemInit** (*Semaphore* *sp, *cnt_t* n)
Initializes a semaphore with the specified counter value.
- void **chSemReset** (*Semaphore* *sp, *cnt_t* n)
Performs a reset operation on the semaphore.
- void **chSemResetl** (*Semaphore* *sp, *cnt_t* n)
Performs a reset operation on the semaphore.
- *msg_t* **chSemWait** (*Semaphore* *sp)
Performs a wait operation on a semaphore.
- *msg_t* **chSemWaitS** (*Semaphore* *sp)
Performs a wait operation on a semaphore.
- *msg_t* **chSemWaitTimeout** (*Semaphore* *sp, *systime_t* time)
Performs a wait operation on a semaphore with timeout specification.
- *msg_t* **chSemWaitTimeoutS** (*Semaphore* *sp, *systime_t* time)
Performs a wait operation on a semaphore with timeout specification.
- void **chSemSignal** (*Semaphore* *sp)
Performs a signal operation on a semaphore.
- void **chSemSignall** (*Semaphore* *sp)
Performs a signal operation on a semaphore.
- void **chSemAddCounterl** (*Semaphore* *sp, *cnt_t* n)
Adds the specified value to the semaphore counter.
- *msg_t* **chSemSignalWait** (*Semaphore* *sp, *Semaphore* *spw)
Performs atomic signal and wait operations on two semaphores.

13.51 chsem.h File Reference

13.51.1 Detailed Description

Semaphores macros and structures.

Data Structures

- struct **Semaphore**
Semaphore structure.

Functions

- void **chSemInit** (*Semaphore* *sp, *cnt_t* n)
Initializes a semaphore with the specified counter value.
- void **chSemReset** (*Semaphore* *sp, *cnt_t* n)
Performs a reset operation on the semaphore.
- void **chSemResetl** (*Semaphore* *sp, *cnt_t* n)

- `msg_t chSemWait (Semaphore *sp)`
Performs a wait operation on a semaphore.
- `msg_t chSemWaitS (Semaphore *sp)`
Performs a wait operation on a semaphore.
- `msg_t chSemWaitTimeout (Semaphore *sp, systime_t time)`
Performs a wait operation on a semaphore with timeout specification.
- `msg_t chSemWaitTimeoutS (Semaphore *sp, systime_t time)`
Performs a wait operation on a semaphore with timeout specification.
- `void chSemSignal (Semaphore *sp)`
Performs a signal operation on a semaphore.
- `void chSemSignall (Semaphore *sp)`
Performs a signal operation on a semaphore.
- `void chSemAddCounterl (Semaphore *sp, cnt_t n)`
Adds the specified value to the semaphore counter.
- `msg_t chSemSignalWait (Semaphore *sp, Semaphore *spw)`
Performs atomic signal and wait operations on two semaphores.

Defines

- `#define _SEMAPHORE_DATA(name, n) {_THREADSQUEUE_DATA(name.s_queue), n}`
Data part of a static semaphore initializer.
- `#define SEMAPHORE_DECL(name, n) Semaphore name = _SEMAPHORE_DATA(name, n)`
Static semaphore initializer.

Macro Functions

- `#define chSemFastWaitl(sp) ((sp)->s_cnt--)`
Decreases the semaphore counter.
- `#define chSemFastSignall(sp) ((sp)->s_cnt++)`
Increases the semaphore counter.
- `#define chSemGetCounterl(sp) ((sp)->s_cnt)`
Returns the semaphore counter current value.

Typedefs

- `typedef struct Semaphore Semaphore`
Semaphore structure.

13.52 chstreams.h File Reference

13.52.1 Detailed Description

Data streams. This header defines abstract interfaces useful to access generic data streams in a standardized way.

Data Structures

- `struct BaseSequentialStreamVMT`
BaseSequentialStream virtual methods table.
- `struct BaseSequentialStream`
Base stream class.

Defines

- `#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.

13.53 chsys.c File Reference

13.53.1 Detailed Description

System related code. `#include "ch.h"`

Functions

- `WORKING_AREA (_idle_thread_wa, PORT_IDLE_THREAD_STACK_SIZE)`
Idle thread working area.
- `void _idle_thread (void *p)`
This function implements the idle thread infinite loop.
- `void chSysInit (void)`
ChibiOS/RT initialization.
- `void chSysTimerHandler1 (void)`
Handles time ticks for round robin preemption and timer increments.

13.54 chsys.h File Reference

13.54.1 Detailed Description

System related macros and structures.

Functions

- `void chSysInit (void)`
ChibiOS/RT initialization.
- `void chSysTimerHandler1 (void)`
Handles time ticks for round robin preemption and timer increments.

Defines

Macro Functions

- `#define chSysGetIdleThread()` (`rlist.r_queue.p_prev`)

Returns a pointer to the idle thread.
- `#define chSysHalt()` `port_halt()`

Halts the system.
- `#define chSysSwitch(ntp, otp)`

Performs a context switch.
- `#define chSysDisable()`

Raises the system interrupt priority mask to the maximum level.
- `#define chSysSuspend()`

Raises the system interrupt priority mask to system level.
- `#define chSysEnable()`

Lowers the system interrupt priority mask to user level.
- `#define chSysLock()`

Enters the kernel lock mode.
- `#define chSysUnlock()`

Leaves the kernel lock mode.
- `#define chSysLockFromIsr()`

Enters the kernel lock mode from within an interrupt handler.
- `#define chSysUnlockFromIsr()`

Leaves the kernel lock mode from within an interrupt handler.

ISRs abstraction macros

- `#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.

13.55 chthreads.c File Reference

13.55.1 Detailed Description

Threads code. `#include "ch.h"`

Functions

- `Thread * _thread_init(Thread *tp, tprio_t prio)`

Initializes a thread structure.
- `void _thread_memfill(uint8_t *startp, uint8_t *endp, uint8_t v)`

Memory fill utility.
- `Thread * chThdCreate1(void *wsp, size_t size, tprio_t prio, tfunc_t pf, void *arg)`

Creates a new thread into a static memory area.
- `Thread * chThdCreateStatic(void *wsp, size_t size, tprio_t prio, tfunc_t pf, void *arg)`

Creates a new thread into a static memory area.

- `tprio_t chThdSetPriority (tprio_t newprio)`
Changes the running thread priority level then reschedules if necessary.
- `Thread * chThdResume (Thread *tp)`
Resumes a suspended thread.
- `void chThdTerminate (Thread *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.
- `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 *tp)`
Blocks the execution of the invoking thread until the specified thread terminates then the exit code is returned.

13.56 chthreads.h File Reference

13.56.1 Detailed Description

Threads macros and structures.

Data Structures

- `struct Thread`
Structure representing a thread.

Functions

- `Thread * _thread_init (Thread *tp, tprio_t prio)`
Initializes a thread structure.
- `void _thread_memfill (uint8_t *startp, uint8_t *endp, uint8_t v)`
Memory fill utility.
- `Thread * chThdCreate1 (void *wsp, size_t size, tprio_t prio, tfunc_t pf, void *arg)`
Creates a new thread into a static memory area.
- `Thread * chThdCreateStatic (void *wsp, size_t size, tprio_t prio, tfunc_t pf, void *arg)`
Creates a new thread into a static memory area.
- `tprio_t chThdSetPriority (tprio_t newprio)`
Changes the running thread priority level then reschedules if necessary.
- `Thread * chThdResume (Thread *tp)`
Resumes a suspended thread.
- `void chThdTerminate (Thread *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.
- 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` *tp)
Blocks the execution of the invoking thread until the specified thread terminates then the exit code is returned.

Defines

Thread states

- `#define THD_STATE_READY` 0
Waiting on the ready list.
- `#define THD_STATE_CURRENT` 1
Currently running.
- `#define THD_STATE_SUSPENDED` 2
Created in suspended state.
- `#define THD_STATE_WTSEM` 3
Waiting on a semaphore.
- `#define THD_STATE_WTMutex` 4
Waiting on a mutex.
- `#define THD_STATE_WTCOND` 5
Waiting on a condition variable.
- `#define THD_STATE_SLEEPING` 6
Waiting in `chThdSleep()` or `chThdSleepUntil()`.
- `#define THD_STATE_WTEXIT` 7
Waiting in `chThdWait()`.
- `#define THD_STATE_WTOREVT` 8
Waiting for an event.
- `#define THD_STATE_WTANDEVT` 9
Waiting for several events.
- `#define THD_STATE SNDMSGQ` 10
Sending a message, in queue.
- `#define THD_STATE SNDMSG` 11
Sent a message, waiting answer.
- `#define THD_STATE_WTMSG` 12
Waiting for a message.
- `#define THD_STATE_WTQUEUE` 13
Waiting on an I/O queue.
- `#define THD_STATE_FINAL` 14
Thread terminated.
- `#define THD_STATE_NAMES`
Thread states as array of strings.

Thread flags and attributes

- `#define THD_MEM_MODE_MASK` 3
Thread memory mode mask.
- `#define THD_MEM_MODE_STATIC` 0
Static thread.
- `#define THD_MEM_MODE_HEAP` 1
Thread allocated from a Memory Heap.
- `#define THD_MEM_MODE_MEMPOOL` 2
Thread allocated from a Memory Pool.
- `#define THD_TERMINATE` 4
Termination requested flag.

Macro Functions

- `#define chThdSelf()` currp
Returns a pointer to the current Thread.
- `#define chThdGetPriority()` (currp->p_prio)
Returns the current thread priority.
- `#define chThdGetTicks(tp) ((tp)->p_time)`
Returns the number of ticks consumed by the specified thread.
- `#define chThdLSS()` (void *)(currp + 1)
Returns the pointer to the Thread local storage area, if any.
- `#define chThdTerminated(tp) ((tp)->p_state == THD_STATE_FINAL)`
Verifies if the specified thread is in the THD_STATE_FINAL state.
- `#define chThdShouldTerminate()` (currp->p_flags & THD_TERMINATE)
Verifies if the current thread has a termination request pending.
- `#define chThdResume(tp) chSchReadyI(tp)`
Resumes a thread created with chThdCreateI().
- `#define chThdSleepS(time) chSchGoSleepTimeoutS(THD_STATE_SLEEPING, time)`
Suspends the invoking thread for the specified time.
- `#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 msg_t(* tfunc_t)(void *)`
Thread function.

13.57 chtypes.h File Reference

13.57.1 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.

```
#include <stddef.h>
#include <stdint.h>
```

Defines

- `#define INLINE inline`
Inline function modifier.
- `#define ROMCONST const`
ROM constant modifier.
- `#define PACK_STRUCT_STRUCT __attribute__((packed))`
Packed structure modifier (within).
- `#define PACK_STRUCT_BEGIN`
Packed structure modifier (before).
- `#define PACK_STRUCT_END`
Packed structure modifier (after).

Typedefs

- `typedef int32_t bool_t`
Boolean, recommended the fastest signed.
- `typedef uint8_t tmode_t`
Thread mode flags, uint8_t is ok.
- `typedef uint8_t tstate_t`
Thread state, uint8_t is ok.
- `typedef uint8_t trefs_t`
Thread references counter, uint8_t is ok.
- `typedef uint32_t tprio_t`
Priority, use the fastest unsigned type.
- `typedef int32_t msg_t`
Message, use signed pointer equivalent.
- `typedef int32_t eventid_t`
Event Id, use fastest signed.
- `typedef uint32_t eventmask_t`
Event Mask, recommended fastest unsigned.
- `typedef uint32_t systime_t`
System Time, recommended fastest unsigned.
- `typedef int32_t cnt_t`
Counter, recommended fastest signed.

13.58 chvt.c File Reference

13.58.1 Detailed Description

Time and Virtual Timers related code. #include "ch.h"

Functions

- `void _vt_init (void)`
Virtual Timers initialization.
- `void chVTSel (VirtualTimer *vtp, systime_t time, vtfunc_t vtfunc, void *par)`
Enables a virtual timer.
- `void chVTResetl (VirtualTimer *vtp)`
Disables a Virtual Timer.

Variables

- `VTLList vlist`
Virtual timers delta list header.

13.59 chvt.h File Reference

13.59.1 Detailed Description

Time macros and structures.

Data Structures

- struct [VirtualTimer](#)
Virtual Timer descriptor structure.
- struct [VTLList](#)
Virtual timers list header.

Functions

- void [_vt_init](#) (void)
Virtual Timers initialization.
- void [chVTSetl](#) ([VirtualTimer](#) *vtp, [systime_t](#) time, [vfunc_t](#) vfunc, void *par)
Enables a virtual timer.
- void [chVTResetl](#) ([VirtualTimer](#) *vtp)
Disables a Virtual Timer.

Variables

- [VTLList](#) vtlist
Virtual timers delta list header.

Defines

Time conversion utilities

- #define [S2ST](#)(sec) (([systime_t](#))((sec) * CH_FREQUENCY))
Seconds to system ticks.
- #define [MS2ST](#)(msec)
Milliseconds to system ticks.
- #define [US2ST](#)(usec)
Microseconds to system ticks.

Macro Functions

- #define [chVTDotickl](#)()
Virtual timers ticker.
- #define [chVTIsArmedl](#)(vtp) ((vtp)->vt_func != NULL)
Returns TRUE if the specified timer is armed.
- #define [chVTSet](#)(vtp, time, vfunc, par)
Enables a virtual timer.
- #define [chVTReset](#)(vtp)
Disables a Virtual Timer.
- #define [chTimeNow](#)() (vtlist.vt_systime)
Current system time.
- #define [chTimeElapsedSince](#)(start) (chTimeNow() - (start))
Returns the elapsed time since the specified start time.
- #define [chTimelsWithin](#)(start, end) (chTimeElapsedSince(start) < ((end) - (start)))
Checks if the current system time is within the specified time window.

Typedefs

- typedef void(* [vfunc_t](#))(void *)
Virtual Timer callback function.
- typedef struct [VirtualTimer](#) [VirtualTimer](#)
Virtual Timer structure type.

13.60 evtimer.c File Reference

13.60.1 Detailed Description

Events Generator Timer code.

```
#include "ch.h"  
#include "evtimer.h"
```

Functions

- void **evtStart** (**EvTimer** *etp)
Starts the timer.
- void **evtStop** (**EvTimer** *etp)
Stops the timer.

13.61 evtimer.h File Reference

13.61.1 Detailed Description

Events Generator Timer structures and macros.

Data Structures

- struct **EvTimer**
Event timer structure.

Functions

- void **evtStart** (**EvTimer** *etp)
Starts the timer.
- void **evtStop** (**EvTimer** *etp)
Stops the timer.

Defines

- #define **evtInit**(etp, time)
*Initializes an **EvTimer** structure.*

13.62 ext.c File Reference

13.62.1 Detailed Description

EXT Driver code.

```
#include "ch.h"  
#include "hal.h"
```

Functions

- void `extInit` (void)
EXT Driver initialization.
- void `extObjectInit` (EXTDriver *extp)
Initializes the standard part of a `EXTDriver` structure.
- void `extStart` (EXTDriver *extp, const EXTConfig *config)
Configures and activates the EXT peripheral.
- void `extStop` (EXTDriver *extp)
Deactivates the EXT peripheral.
- void `extChannelEnable` (EXTDriver *extp, expchannel_t channel)
Enables an EXT channel.
- void `extChannelDisable` (EXTDriver *extp, expchannel_t channel)
Disables an EXT channel.
- void `extSetChannelModel` (EXTDriver *extp, expchannel_t channel, const EXTChannelConfig *extcp)
Changes the operation mode of a channel.

13.63 ext.h File Reference

13.63.1 Detailed Description

EXT Driver macros and structures. #include "ext_lld.h"

Functions

- void `extInit` (void)
EXT Driver initialization.
- void `extObjectInit` (EXTDriver *extp)
Initializes the standard part of a `EXTDriver` structure.
- void `extStart` (EXTDriver *extp, const EXTConfig *config)
Configures and activates the EXT peripheral.
- void `extStop` (EXTDriver *extp)
Deactivates the EXT peripheral.
- void `extChannelEnable` (EXTDriver *extp, expchannel_t channel)
Enables an EXT channel.
- void `extChannelDisable` (EXTDriver *extp, expchannel_t channel)
Disables an EXT channel.
- void `extSetChannelModel` (EXTDriver *extp, expchannel_t channel, const EXTChannelConfig *extcp)
Changes the operation mode of a channel.

Defines

EXT channel modes

- #define `EXT_CH_MODE_EDGES_MASK` 3
Mask of edges field.
- #define `EXT_CH_MODE_DISABLED` 0
Channel disabled.
- #define `EXT_CH_MODE_RISING_EDGE` 1
Rising edge callback.
- #define `EXT_CH_MODE_FALLING_EDGE` 2

- `#define EXT_CH_MODE_BOTH_EDGES 3`
Both edges callback.
- `#define EXT_CH_MODE_AUTOSTART 4`
Channel started automatically on driver start.

Macro Functions

- `#define extChannelEnable(extp, channel) ext_lld_channel_enable(extp, channel)`
Enables an EXT channel.
- `#define extChannelDisable(extp, channel) ext_lld_channel_disable(extp, channel)`
Disables an EXT channel.
- `#define extSetChannelMode(extp, channel, extcp)`
Changes the operation mode of a channel.

Typedefs

- `typedef struct EXTDriver EXTDriver`
Type of a structure representing a EXT driver.

Enumerations

- `enum extstate_t { EXT_UNINIT = 0, EXT_STOP = 1, EXT_ACTIVE = 2 }`
Driver state machine possible states.

13.64 ext_lld.c File Reference

13.64.1 Detailed Description

EXT Driver subsystem low level driver source template.

```
#include "ch.h"
#include "hal.h"
```

Functions

- `void ext_lld_init (void)`
Low level EXT driver initialization.
- `void ext_lld_start (EXTDriver *extp)`
Configures and activates the EXT peripheral.
- `void ext_lld_stop (EXTDriver *extp)`
Deactivates the EXT peripheral.
- `void ext_lld_channel_enable (EXTDriver *extp, expchannel_t channel)`
Enables an EXT channel.
- `void ext_lld_channel_disable (EXTDriver *extp, expchannel_t channel)`
Disables an EXT channel.

Variables

- `EXTDriver EXTD1`
EXT1 driver identifier.

13.65 ext_ll.h File Reference

13.65.1 Detailed Description

EXT Driver subsystem low level driver header template.

Data Structures

- struct `EXTChannelConfig`
Channel configuration structure.
- struct `EXTConfig`
Driver configuration structure.
- struct `EXTDriver`
Structure representing an EXT driver.

Functions

- void `ext_ll_init` (void)
Low level EXT driver initialization.
- void `ext_ll_start` (`EXTDriver` *`extp`)
Configures and activates the EXT peripheral.
- void `ext_ll_stop` (`EXTDriver` *`extp`)
Deactivates the EXT peripheral.
- void `ext_ll_channel_enable` (`EXTDriver` *`extp`, `expchannel_t` `channel`)
Enables an EXT channel.
- void `ext_ll_channel_disable` (`EXTDriver` *`extp`, `expchannel_t` `channel`)
Disables an EXT channel.

Defines

- #define `EXT_MAX_CHANNELS` 20
Available number of EXT channels.

Configuration options

- #define `PLATFORM_EXT_USE_EXT1` FALSE
EXT driver enable switch.

TypeDefs

- typedef `uint32_t` `expchannel_t`
EXT channel identifier.
- typedef void(* `extcallback_t`)(`EXTDriver` *`extp`, `expchannel_t` `channel`)
Type of an EXT generic notification callback.

13.66 gpt.c File Reference

13.66.1 Detailed Description

```
GPT Driver code. #include "ch.h"
#include "hal.h"
```

Functions

- void `gptInit` (void)

GPT Driver initialization.
- void `gptObjectInit` (`GPTDriver` *`gptp`)

Initializes the standard part of a `GPTDriver` structure.
- void `gptStart` (`GPTDriver` *`gptp`, const `GPTConfig` *`config`)

Configures and activates the GPT peripheral.
- void `gptStop` (`GPTDriver` *`gptp`)

Deactivates the GPT peripheral.
- void `gptChangeInterval` (`GPTDriver` *`gptp`, `gptcnt_t` `interval`)

Changes the interval of GPT peripheral.
- void `gptStartContinuous` (`GPTDriver` *`gptp`, `gptcnt_t` `interval`)

Starts the timer in continuous mode.
- void `gptStartContinuousl` (`GPTDriver` *`gptp`, `gptcnt_t` `interval`)

Starts the timer in continuous mode.
- void `gptStartOneShot` (`GPTDriver` *`gptp`, `gptcnt_t` `interval`)

Starts the timer in one shot mode.
- void `gptStartOneShotl` (`GPTDriver` *`gptp`, `gptcnt_t` `interval`)

Starts the timer in one shot mode.
- void `gptStopTimer` (`GPTDriver` *`gptp`)

Stops the timer.
- void `gptStopTimerl` (`GPTDriver` *`gptp`)

Stops the timer.
- void `gptPolledDelay` (`GPTDriver` *`gptp`, `gptcnt_t` `interval`)

Starts the timer in one shot mode and waits for completion.

13.67 gpt.h File Reference

13.67.1 Detailed Description

GPT Driver macros and structures. #include "gpt_lld.h"

Functions

- void `gptInit` (void)

GPT Driver initialization.
- void `gptObjectInit` (`GPTDriver` *`gptp`)

Initializes the standard part of a `GPTDriver` structure.
- void `gptStart` (`GPTDriver` *`gptp`, const `GPTConfig` *`config`)

Configures and activates the GPT peripheral.
- void `gptStop` (`GPTDriver` *`gptp`)

Deactivates the GPT peripheral.
- void `gptStartContinuous` (`GPTDriver` *`gptp`, `gptcnt_t` `interval`)

Starts the timer in continuous mode.
- void `gptStartContinuousl` (`GPTDriver` *`gptp`, `gptcnt_t` `interval`)

Starts the timer in continuous mode.
- void `gptChangeInterval` (`GPTDriver` *`gptp`, `gptcnt_t` `interval`)

Changes the interval of GPT peripheral.
- void `gptStartOneShot` (`GPTDriver` *`gptp`, `gptcnt_t` `interval`)

- void `gptStartOneShotl` (`GPTDriver` *`gptp`, `gptcnt_t` `interval`)
Starts the timer in one shot mode.
- void `gptStopTimer` (`GPTDriver` *`gptp`)
Stops the timer.
- void `gptStopTimerl` (`GPTDriver` *`gptp`)
Stops the timer.
- void `gptPolledDelay` (`GPTDriver` *`gptp`, `gptcnt_t` `interval`)
Starts the timer in one shot mode and waits for completion.

Defines

- #define `gptChangeInterval`(`gptp`, `interval`)
Changes the interval of GPT peripheral.

Typedefs

- typedef struct `GPTDriver` `GPTDriver`
Type of a structure representing a GPT driver.
- typedef void(* `gptcallback_t`)(`GPTDriver` *`gptp`)
GPT notification callback type.

Enumerations

- enum `gptstate_t` {
`GPT_UNINIT` = 0, `GPT_STOP` = 1, `GPT_READY` = 2, `GPT_CONTINUOUS` = 3,
`GPT_ONESHOT` = 4 }
Driver state machine possible states.

13.68 gpt_lld.c File Reference

13.68.1 Detailed Description

GPT Driver subsystem low level driver source template.

```
#include "ch.h"
#include "hal.h"
```

Functions

- void `gpt_lld_init` (void)
Low level GPT driver initialization.
- void `gpt_lld_start` (`GPTDriver` *`gptp`)
Configures and activates the GPT peripheral.
- void `gpt_lld_stop` (`GPTDriver` *`gptp`)
Deactivates the GPT peripheral.
- void `gpt_lld_start_timer` (`GPTDriver` *`gptp`, `gptcnt_t` `interval`)
Starts the timer in continuous mode.
- void `gpt_lld_stop_timer` (`GPTDriver` *`gptp`)
Stops the timer.
- void `gpt_lld_polled_delay` (`GPTDriver` *`gptp`, `gptcnt_t` `interval`)
Starts the timer in one shot mode and waits for completion.

Variables

- **GPTDriver GPTD1**
GPTD1 driver identifier.

13.69 gpt_ll.h File Reference

13.69.1 Detailed Description

GPT Driver subsystem low level driver header template.

Data Structures

- struct **GPTConfig**
Driver configuration structure.
- struct **GPTDriver**
Structure representing a GPT driver.

Functions

- void **gpt_ll_init** (void)
Low level GPT driver initialization.
- void **gpt_ll_start** (GPTDriver *gptp)
Configures and activates the GPT peripheral.
- void **gpt_ll_stop** (GPTDriver *gptp)
Deactivates the GPT peripheral.
- void **gpt_ll_start_timer** (GPTDriver *gptp, gptcnt_t interval)
Starts the timer in continuous mode.
- void **gpt_ll_stop_timer** (GPTDriver *gptp)
Stops the timer.
- void **gpt_ll_polled_delay** (GPTDriver *gptp, gptcnt_t interval)
Starts the timer in one shot mode and waits for completion.

Defines

- #define **gpt_ll_change_interval**(gptp, interval)
Changes the interval of GPT peripheral.

Configuration options

- #define **STM32_GPT_USE_TIM1** FALSE
GPTD1 driver enable switch.

TypeDefs

- typedef uint32_t **gptfreq_t**
GPT frequency type.
- typedef uint16_t **gptcnt_t**
GPT counter type.

13.70 hal.c File Reference

13.70.1 Detailed Description

```
HAL subsystem code. #include "ch.h"  
#include "hal.h"
```

Functions

- void `halInit` (void)
HAL initialization.
- `bool_t hallsCounterWithin` (`halrcnt_t` start, `halrcnt_t` end)
Realtime window test.
- void `halPolledDelay` (`halrcnt_t` ticks)
Polled delay.

13.71 hal.h File Reference

13.71.1 Detailed Description

```
HAL subsystem header. #include "ch.h"  
#include "board.h"  
#include "halconf.h"  
#include "hal_lld.h"  
#include "io_channel.h"  
#include "io_block.h"  
#include "mmcsd.h"  
#include "tm.h"  
#include "pal.h"  
#include "adc.h"  
#include "can.h"  
#include "ext.h"  
#include "gpt.h"  
#include "i2c.h"  
#include "icu.h"  
#include "mac.h"  
#include "pwm.h"  
#include "rtc.h"  
#include "serial.h"  
#include "sdc.h"  
#include "spi.h"  
#include "uart.h"  
#include "usb.h"
```

```
#include "mmc_spi.h"
#include "serial_usb.h"
```

Functions

- void **halInit** (void)

HAL initialization.

Defines

Time conversion utilities for the realtime counter

- #define **S2RTT**(sec) (halGetCounterFrequency() * (sec))

Seconds to realtime ticks.
- #define **MS2RTT**(msec) (((halGetCounterFrequency() + 999UL) / 1000UL) * (msec))

Milliseconds to realtime ticks.
- #define **US2RTT**(usec)

Microseconds to realtime ticks.
- #define **RTT2S**(ticks) ((ticks) / halGetCounterFrequency())

Realtime ticks to seconds to.
- #define **RTT2MS**(ticks) ((ticks) / (halGetCounterFrequency() / 1000UL))

Realtime ticks to milliseconds.
- #define **RTT2US**(ticks) ((ticks) / (halGetCounterFrequency() / 1000000UL))

Realtime ticks to microseconds.

Macro Functions

- #define **halGetCounterValue**() hal_lld_get_counter_value()

Returns the current value of the system free running counter.
- #define **halGetCounterFrequency**() hal_lld_get_counter_frequency()

Realtime counter frequency.

13.72 hal_lld.c File Reference

13.72.1 Detailed Description

HAL Driver subsystem low level driver source template.

```
#include "ch.h"
#include "hal.h"
```

Functions

- void **hal_lld_init** (void)

Low level HAL driver initialization.
- void **platform_early_init** (void)

Platform early initialization.

13.73 hal_lld.h File Reference

13.73.1 Detailed Description

HAL subsystem low level driver header template.

Functions

- void `hal_lld_init` (void)
Low level HAL driver initialization.

Defines

- `#define HAL_IMPLEMENTS_COUNTERS TRUE`
Defines the support for realtime counters in the HAL.
- `#define hal_lld_get_counter_value() 0`
Returns the current value of the system free running counter.
- `#define hal_lld_get_counter_frequency() 0`
Realtime counter frequency.

Platform identification

- `#define PLATFORM_NAME ""`

Typedefs

- `typedef uint32_t halclock_t`
Type representing a system clock frequency.
- `typedef uint32_t halrtcnt_t`
Type of the realtime free counter value.

13.74 halconf.h File Reference

13.74.1 Detailed Description

HAL configuration header. HAL configuration file, this file allows to enable or disable the various device drivers from your application. You may also use this file in order to override the device drivers default settings. `#include "mcuconf.h"`

Defines

Drivers enable switches

- `#define HAL_USE_TM TRUE`
Enables the TM subsystem.
- `#define HAL_USE_PAL TRUE`
Enables the PAL subsystem.
- `#define HAL_USE_ADC TRUE`
Enables the ADC subsystem.
- `#define HAL_USE_CAN TRUE`
Enables the CAN subsystem.
- `#define HAL_USE_EXT TRUE`
Enables the EXT subsystem.
- `#define HAL_USE_GPT TRUE`
Enables the GPT subsystem.
- `#define HAL_USE_I2C TRUE`
Enables the I2C subsystem.
- `#define HAL_USE_ICU TRUE`
Enables the ICU subsystem.

- `#define HAL_USE_MAC TRUE`
Enables the MAC subsystem.
- `#define HAL_USE_MMC_SPI TRUE`
Enables the MMC_SPI subsystem.
- `#define HAL_USE_PWM TRUE`
Enables the PWM subsystem.
- `#define HAL_USE_RTC FALSE`
Enables the RTC subsystem.
- `#define HAL_USE_SDC TRUE`
Enables the SDC subsystem.
- `#define HAL_USE_SERIAL TRUE`
Enables the SERIAL subsystem.
- `#define HAL_USE_SERIAL_USB TRUE`
Enables the SERIAL over USB subsystem.
- `#define HAL_USE_SPI TRUE`
Enables the SPI subsystem.
- `#define HAL_USE_UART TRUE`
Enables the UART subsystem.
- `#define HAL_USE_USB TRUE`
Enables the USB subsystem.

ADC driver related setting

- `#define ADC_USE_WAIT TRUE`
Enables synchronous APIs.
- `#define ADC_USE_MUTUAL_EXCLUSION TRUE`
Enables the `adcAcquireBus()` and `adcReleaseBus()` APIs.

CAN driver related setting

- `#define CAN_USE_SLEEP_MODE TRUE`
Sleep mode related APIs inclusion switch.

I2C driver related setting

- `#define I2C_USE_MUTUAL_EXCLUSION TRUE`
Enables the mutual exclusion APIs on the I2C bus.

MAC driver related setting

- `#define MAC_USE_ZERO_COPY TRUE`
Enables an event sources for incoming packets.
- `#define MAC_USE_EVENTS TRUE`
Enables an event sources for incoming packets.

MMC_SPI driver related setting

- `#define MMC_NICE_WAITING TRUE`
Delays insertions.

SDC driver related setting

- `#define SDC_INIT_RETRY 100`
Number of initialization attempts before rejecting the card.
- `#define SDC_MMC_SUPPORT TRUE`
Include support for MMC cards.
- `#define SDC_NICE_WAITING TRUE`
Delays insertions.

SERIAL driver related setting

- #define **SERIAL_DEFAULT_BITRATE** 38400
Default bit rate.
- #define **SERIAL_BUFFERS_SIZE** 16
Serial buffers size.

SERIAL_USB driver related setting

- #define **SERIAL_USB_BUFFERS_SIZE** 64
Serial over USB buffers size.

SPI driver related setting

- #define **SPI_USE_WAIT** TRUE
Enables synchronous APIs.
- #define **SPI_USE_MUTUAL_EXCLUSION** TRUE
Enables the `spiAcquireBus()` and `spiReleaseBus()` APIs.

13.75 i2c.c File Reference**13.75.1 Detailed Description**

I2C Driver code.

```
#include "ch.h"
#include "hal.h"
```

Functions

- void **i2cInit** (void)
I2C Driver initialization.
- void **i2cObjectInit** (**I2CDriver** *i2cp)
Initializes the standard part of a `I2CDriver` structure.
- void **i2cStart** (**I2CDriver** *i2cp, const **I2CConfig** *config)
Configures and activates the I2C peripheral.
- void **i2cStop** (**I2CDriver** *i2cp)
Deactivates the I2C peripheral.
- **i2cflags_t** **i2cGetErrors** (**I2CDriver** *i2cp)
Returns the errors mask associated to the previous operation.
- **msg_t** **i2cMasterTransmitTimeout** (**I2CDriver** *i2cp, **i2caddr_t** addr, const **uint8_t** *txbuf, **size_t** txbytes, **uint8_t** *rxbuf, **size_t** rxbytes, **systime_t** timeout)
Sends data via the I2C bus.
- **msg_t** **i2cMasterReceiveTimeout** (**I2CDriver** *i2cp, **i2caddr_t** addr, **uint8_t** *rxbuf, **size_t** rxbytes, **systime_t** timeout)
Receives data from the I2C bus.
- void **i2cAcquireBus** (**I2CDriver** *i2cp)
Gains exclusive access to the I2C bus.
- void **i2cReleaseBus** (**I2CDriver** *i2cp)
Releases exclusive access to the I2C bus.

13.76 i2c.h File Reference**13.76.1 Detailed Description**

I2C Driver macros and structures.

```
#include "i2c_lld.h"
```

Functions

- void `i2cInit (void)`
I2C Driver initialization.
- void `i2cObjectInit (I2CDriver *i2cp)`
Initializes the standard part of a `I2CDriver` structure.
- void `i2cStart (I2CDriver *i2cp, const I2CConfig *config)`
Configures and activates the I2C peripheral.
- void `i2cStop (I2CDriver *i2cp)`
Deactivates the I2C peripheral.
- `i2cflags_t i2cGetErrors (I2CDriver *i2cp)`
Returns the errors mask associated to the previous operation.
- `msg_t i2cMasterTransmitTimeout (I2CDriver *i2cp, i2caddr_t addr, const uint8_t *txbuf, size_t txbytes, uint8_t *rxbuf, size_t rxbytes, systime_t timeout)`
Sends data via the I2C bus.
- `msg_t i2cMasterReceiveTimeout (I2CDriver *i2cp, i2caddr_t addr, uint8_t *rxbuf, size_t rxbytes, systime_t timeout)`
Receives data from the I2C bus.

Defines

- `#define I2C_USE_MUTUAL_EXCLUSION TRUE`
Enables the mutual exclusion APIs on the I2C bus.
- `#define i2cMasterTransmit(i2cp, addr, txbuf, txbytes, rxbuf, rxbytes)`
Wrap `i2cMasterTransmitTimeout` function with `TIME_INFINITE` timeout.
- `#define i2cMasterReceive(i2cp, addr, rxbuf, rxbytes) (i2cMasterReceiveTimeout(i2cp, addr, rxbuf, rxbytes, TIME_INFINITE))`
Wrap `i2cMasterReceiveTimeout` function with `TIME_INFINITE` timeout.

I2C bus error conditions

- `#define I2CD_NO_ERROR 0x00`
No error.
- `#define I2CD_BUS_ERROR 0x01`
Bus Error.
- `#define I2CD_ARBITRATION_LOST 0x02`
Arbitration Lost.
- `#define I2CD_ACK_FAILURE 0x04`
Acknowledge Failure.
- `#define I2CD_OVERRUN 0x08`
Overrun/Underrun.
- `#define I2CD_PEC_ERROR 0x10`
PEC Error in reception.
- `#define I2CD_TIMEOUT 0x20`
Hardware timeout.
- `#define I2CD_SMB_ALERT 0x40`
SMBus Alert.

Enumerations

- enum `i2cstate_t {`
`I2C_UNINIT = 0, I2C_STOP = 1, I2C_READY = 2, I2C_ACTIVE_TX = 3,`
`I2C_ACTIVE_RX = 4 }`
Driver state machine possible states.

13.77 i2c_lld.c File Reference

13.77.1 Detailed Description

I2C Driver subsystem low level driver source template.

```
#include "ch.h"
#include "hal.h"
```

Functions

- void **i2c_lld_init** (void)
Low level I2C driver initialization.
- void **i2c_lld_start** (I2CDriver *i2cp)
Configures and activates the I2C peripheral.
- void **i2c_lld_stop** (I2CDriver *i2cp)
Deactivates the I2C peripheral.
- msg_t **i2c_lld_master_receive_timeout** (I2CDriver *i2cp, i2caddr_t addr, uint8_t *rdbuf, size_t rxbytes, systime_t timeout)
Receives data via the I2C bus as master.
- msg_t **i2c_lld_master_transmit_timeout** (I2CDriver *i2cp, i2caddr_t addr, const uint8_t *txbuf, size_t txbytes, uint8_t *rdbuf, size_t rxbytes, systime_t timeout)
Transmits data via the I2C bus as master.

Variables

- I2CDriver **I2CD1**
I2C1 driver identifier.

13.78 i2c_lld.h File Reference

13.78.1 Detailed Description

I2C Driver subsystem low level driver header template.

Data Structures

- struct **I2CConfig**
Driver configuration structure.
- struct **I2CDriver**
Structure representing an I2C driver.

Functions

- void **i2c_lld_init** (void)
Low level I2C driver initialization.
- void **i2c_lld_start** (I2CDriver *i2cp)
Configures and activates the I2C peripheral.
- void **i2c_lld_stop** (I2CDriver *i2cp)
Deactivates the I2C peripheral.

- `msg_t i2c_lld_master_transmit_timeout (I2CDriver *i2cp, i2caddr_t addr, const uint8_t *txbuf, size_t txbytes, uint8_t *rdbuf, size_t rxbytes, systime_t timeout)`
Transmits data via the I2C bus as master.
- `msg_t i2c_lld_master_receive_timeout (I2CDriver *i2cp, i2caddr_t addr, uint8_t *rdbuf, size_t rxbytes, systime_t timeout)`
Receives data via the I2C bus as master.

Defines

- `#define i2c_lld_get_errors(i2cp) ((i2cp)->errors)`
Get errors from I2C driver.

Configuration options

- `#define PLATFORM_I2C_USE_I2C1 FALSE`
I2C1 driver enable switch.

Typedefs

- `typedef uint16_t i2caddr_t`
Type representing I2C address.
- `typedef uint32_t i2cflags_t`
Type of I2C Driver condition flags.
- `typedef struct I2CDriver I2CDriver`
Type of a structure representing an I2C driver.

13.79 icu.c File Reference

13.79.1 Detailed Description

```
ICU Driver code. #include "ch.h"
#include "hal.h"
```

Functions

- `void icuInit (void)`
ICU Driver initialization.
- `void icuObjectInit (ICUDriver *icup)`
Initializes the standard part of a `ICUDriver` structure.
- `void icuStart (ICUDriver *icup, const ICUConfig *config)`
Configures and activates the ICU peripheral.
- `void icuStop (ICUDriver *icup)`
Deactivates the ICU peripheral.
- `void icuEnable (ICUDriver *icup)`
Enables the input capture.
- `void icuDisable (ICUDriver *icup)`
Disables the input capture.

13.80 icu.h File Reference

13.80.1 Detailed Description

ICU Driver macros and structures. #include "icu_lld.h"

Functions

- void **icuInit** (void)
ICU Driver initialization.
- void **icuObjectInit** (ICUDriver *icup)
Initializes the standard part of a `ICUDriver` structure.
- void **icuStart** (ICUDriver *icup, const ICUConfig *config)
Configures and activates the ICU peripheral.
- void **icuStop** (ICUDriver *icup)
Deactivates the ICU peripheral.
- void **icuEnable** (ICUDriver *icup)
Enables the input capture.
- void **icuDisable** (ICUDriver *icup)
Disables the input capture.

Defines

Macro Functions

- #define **icuEnable**(icup) icu_lld_enable(icup)
Enables the input capture.
- #define **icuDisable**(icup) icu_lld_disable(icup)
Disables the input capture.
- #define **icuGetWidth**(icup) icu_lld_get_width(icup)
Returns the width of the latest pulse.
- #define **icuGetPeriod**(icup) icu_lld_get_period(icup)
Returns the width of the latest cycle.

Low Level driver helper macros

- #define **_icu_isr_invoke_width_cb**(icup)
Common ISR code, ICU width event.
- #define **_icu_isr_invoke_period_cb**(icup)
Common ISR code, ICU period event.
- #define **_icu_isr_invoke_overflow_cb**(icup)
Common ISR code, ICU timer overflow event.

Typedefs

- typedef struct **ICUDriver** **ICUDriver**
Type of a structure representing an ICU driver.
- typedef void(* **icucallback_t**)(ICUDriver *icup)
ICU notification callback type.

Enumerations

- enum `icustate_t` {

 `ICU_UNINIT` = 0, `ICU_STOP` = 1, `ICU_READY` = 2, `ICU_WAITING` = 3,

 `ICU_ACTIVE` = 4, `ICU_IDLE` = 5 }

Driver state machine possible states.

13.81 icu_lld.c File Reference

13.81.1 Detailed Description

ICU Driver subsystem low level driver source template.

```
#include "ch.h"
#include "hal.h"
```

Functions

- void `icu_lld_init` (void)

Low level ICU driver initialization.
- void `icu_lld_start` (ICUDriver *icup)

Configures and activates the ICU peripheral.
- void `icu_lld_stop` (ICUDriver *icup)

Deactivates the ICU peripheral.
- void `icu_lld_enable` (ICUDriver *icup)

Enables the input capture.
- void `icu_lld_disable` (ICUDriver *icup)

Disables the input capture.
- `icucnt_t icu_lld_get_width` (ICUDriver *icup)

Returns the width of the latest pulse.
- `icucnt_t icu_lld_get_period` (ICUDriver *icup)

Returns the width of the latest cycle.

Variables

- ICUDriver `ICUD1`

ICU1 driver identifier.

13.82 icu_lld.h File Reference

13.82.1 Detailed Description

ICU Driver subsystem low level driver header template.

Data Structures

- struct `ICUConfig`

Driver configuration structure.
- struct `ICUDriver`

Structure representing an ICU driver.

Functions

- void `icu_lld_init` (void)
Low level ICU driver initialization.
- void `icu_lld_start` (ICUDriver *icup)
Configures and activates the ICU peripheral.
- void `icu_lld_stop` (ICUDriver *icup)
Deactivates the ICU peripheral.
- void `icu_lld_enable` (ICUDriver *icup)
Enables the input capture.
- void `icu_lld_disable` (ICUDriver *icup)
Disables the input capture.
- `icucnt_t icu_lld_get_width` (ICUDriver *icup)
Returns the width of the latest pulse.
- `icucnt_t icu_lld_get_period` (ICUDriver *icup)
Returns the width of the latest cycle.

Defines

Configuration options

- `#define PLATFORM_ICU_USE_ICU1 FALSE`
ICU driver enable switch.

Typedefs

- `typedef uint32_t icufreq_t`
ICU frequency type.
- `typedef uint16_t icucnt_t`
ICU counter type.

Enumerations

- enum `icumode_t` { `ICU_INPUT_ACTIVE_HIGH` = 0, `ICU_INPUT_ACTIVE_LOW` = 1 }
ICU driver mode.

13.83 io_block.h File Reference

13.83.1 Detailed Description

I/O block devices access. This header defines an abstract interface useful to access generic I/O block devices in a standardized way.

Data Structures

- struct `BlockDeviceInfo`
Block device info.
- struct `BaseBlockDeviceVMT`
BaseBlockDevice virtual methods table.
- struct `BaseBlockDevice`
Base block device class.

Defines

- `#define _base_block_device_methods`
BaseBlockDevice specific methods.
- `#define _base_block_device_data`
BaseBlockDevice specific data.

Macro Functions (BaseBlockDevice)

- `#define blkGetDriverState(ip) ((ip)->state)`
Returns the driver state.
- `#define blkIsTransferring(ip)`
Determines if the device is transferring data.
- `#define blkIsInserted(ip) ((ip)->vmt->is_inserted(ip))`
Returns the media insertion status.
- `#define blkIsWriteProtected(ip) ((ip)->vmt->is_protected(ip))`
Returns the media write protection status.
- `#define blkConnect(ip) ((ip)->vmt->connect(ip))`
Performs the initialization procedure on the block device.
- `#define blkDisconnect(ip) ((ip)->vmt->disconnect(ip))`
Terminates operations on the block device.
- `#define blkRead(ip, startblk, buf, n) ((ip)->vmt->read(ip, startblk, buf, n))`
Reads one or more blocks.
- `#define blkWrite(ip, startblk, buf, n) ((ip)->vmt->write(ip, startblk, buf, n))`
Writes one or more blocks.
- `#define blkSync(ip) ((ip)->vmt->sync(ip))`
Ensures write synchronization.
- `#define blkGetInfo(ip, bdip) ((ip)->vmt->get_info(ip, bdip))`
Returns a media information structure.

Enumerations

- enum `blkstate_t` {

`BLK_UNINIT = 0, BLK_STOP = 1, BLK_ACTIVE = 2, BLK_CONNECTING = 3,`

`BLK_DISCONNECTING = 4, BLK_READY = 5, BLK_READING = 6, BLK_WRITING = 7,`

`BLK_SYNCING = 8 }`

Driver state machine possible states.

13.84 io_channel.h File Reference

13.84.1 Detailed Description

I/O channels access. This header defines an abstract interface useful to access generic I/O serial devices in a standardized way.

Data Structures

- struct `BaseChannelVMT`
BaseChannel virtual methods table.
- struct `BaseChannel`
Base channel class.
- struct `BaseAsynchronousChannelVMT`
BaseAsynchronousChannel virtual methods table.
- struct `BaseAsynchronousChannel`
Base asynchronous channel class.

Defines

- `#define _base_channel_methods`
BaseChannel specific methods.
- `#define _base_channel_data _base_sequential_stream_data`
BaseChannel specific data.
- `#define _base_asynchronous_channel_methods _base_channel_methods \`
BaseAsynchronousChannel specific methods.
- `#define _base_asynchronous_channel_data`
BaseAsynchronousChannel specific data.

Macro Functions (BaseChannel)

- `#define chnPutTimeout(ip, b, time) ((ip)->vmt->putt(ip, b, time))`
Channel blocking byte write with timeout.
- `#define chnGetTimeout(ip, time) ((ip)->vmt->gett(ip, time))`
Channel blocking byte read with timeout.
- `#define chnWrite(ip, bp, n) chSequentialStreamWrite(ip, bp, n)`
Channel blocking write.
- `#define chnWriteTimeout(ip, bp, n, time) ((ip)->vmt->writet(ip, bp, n, time))`
Channel blocking write with timeout.
- `#define chnRead(ip, bp, n) chSequentialStreamRead(ip, bp, n)`
Channel blocking read.
- `#define chnReadTimeout(ip, bp, n, time) ((ip)->vmt->readt(ip, bp, n, time))`
Channel blocking read with timeout.

I/O status flags added to the event listener

- `#define CHN_NO_ERROR 0`
No pending conditions.
- `#define CHN_CONNECTED 1`
Connection happened.
- `#define CHN_DISCONNECTED 2`
Disconnection happened.
- `#define CHN_INPUT_AVAILABLE 4`
Data available in the input queue.
- `#define CHN_OUTPUT_EMPTY 8`
Output queue empty.
- `#define CHN_TRANSMISSION_END 16`
Transmission end.

Macro Functions (BaseAsynchronousChannel)

- `#define chnGetEventSource(ip) (&((ip)->event))`
Returns the I/O condition event source.
- `#define chnAddFlags1(ip, flags)`
Adds status flags to the listeners's flags mask.

13.85 mac.c File Reference

13.85.1 Detailed Description

MAC Driver code. `#include "ch.h"`

```
#include "hal.h"
```

Functions

- void `macInit` (void)
MAC Driver initialization.
- void `macObjectInit` (`MACDriver` *`macp`)
Initialize the standard part of a `MACDriver` structure.
- void `macStart` (`MACDriver` *`macp`, const `MACConfig` *`config`)
Configures and activates the MAC peripheral.
- void `macStop` (`MACDriver` *`macp`)
Deactivates the MAC peripheral.
- `msg_t` `macWaitTransmitDescriptor` (`MACDriver` *`macp`, `MACTransmitDescriptor` *`tdp`, `systime_t` `time`)
Allocates a transmission descriptor.
- void `macReleaseTransmitDescriptor` (`MACTransmitDescriptor` *`tdp`)
Releases a transmit descriptor and starts the transmission of the enqueued data as a single frame.
- `msg_t` `macWaitReceiveDescriptor` (`MACDriver` *`macp`, `MACReceiveDescriptor` *`rdp`, `systime_t` `time`)
Waits for a received frame.
- void `macReleaseReceiveDescriptor` (`MACReceiveDescriptor` *`rdp`)
Releases a receive descriptor.
- `bool_t` `macPollLinkStatus` (`MACDriver` *`macp`)
Updates and returns the link status.

13.86 mac.h File Reference

13.86.1 Detailed Description

MAC Driver macros and structures. #include "mac_lld.h"

Functions

- void `macInit` (void)
MAC Driver initialization.
- void `macObjectInit` (`MACDriver` *`macp`)
Initialize the standard part of a `MACDriver` structure.
- void `macStart` (`MACDriver` *`macp`, const `MACConfig` *`config`)
Configures and activates the MAC peripheral.
- void `macStop` (`MACDriver` *`macp`)
Deactivates the MAC peripheral.
- `msg_t` `macWaitTransmitDescriptor` (`MACDriver` *`macp`, `MACTransmitDescriptor` *`tdp`, `systime_t` `time`)
Allocates a transmission descriptor.
- void `macReleaseTransmitDescriptor` (`MACTransmitDescriptor` *`tdp`)
Releases a transmit descriptor and starts the transmission of the enqueued data as a single frame.
- `msg_t` `macWaitReceiveDescriptor` (`MACDriver` *`macp`, `MACReceiveDescriptor` *`rdp`, `systime_t` `time`)
Waits for a received frame.
- void `macReleaseReceiveDescriptor` (`MACReceiveDescriptor` *`rdp`)
Releases a receive descriptor.
- `bool_t` `macPollLinkStatus` (`MACDriver` *`macp`)
Updates and returns the link status.

Defines

MAC configuration options

- #define `MAC_USE_ZERO_COPY` FALSE
Enables an event sources for incoming packets.
- #define `MAC_USE_EVENTS` TRUE
Enables an event sources for incoming packets.

Macro Functions

- #define `macGetReceiveEventSource(macp)` (&(macp)->rdevent)
Returns the received frames event source.
- #define `macWriteTransmitDescriptor(tdp, buf, size)` `mac_lld_write_transmit_descriptor(tdp, buf, size)`
Writes to a transmit descriptor's stream.
- #define `macReadReceiveDescriptor(rdp, buf, size)` `mac_lld_read_receive_descriptor(rdp, buf, size)`
Reads from a receive descriptor's stream.
- #define `macGetNextTransmitBuffer(tdp, size, sizep)` `mac_lld_get_next_transmit_buffer(tdp, size, sizep)`
Returns a pointer to the next transmit buffer in the descriptor chain.
- #define `macGetNextReceiveBuffer(rdp, sizep)` `mac_lld_get_next_receive_buffer(rdp, sizep)`
Returns a pointer to the next receive buffer in the descriptor chain.

Typedefs

- typedef struct `MACDriver` `MACDriver`
Type of a structure representing a MAC driver.

Enumerations

- enum `macstate_t` { `MAC_UNINIT` = 0, `MAC_STOP` = 1, `MAC_ACTIVE` = 2 }
Driver state machine possible states.

13.87 mac_lld.c File Reference

13.87.1 Detailed Description

MAC Driver subsystem low level driver source template.

```
#include <string.h>
#include "ch.h"
#include "hal.h"
#include "mii.h"
```

Functions

- void `mac_lld_init` (void)
Low level MAC initialization.
- void `mac_lld_start` (`MACDriver` *macp)
Configures and activates the MAC peripheral.
- void `mac_lld_stop` (`MACDriver` *macp)
Deactivates the MAC peripheral.
- `msg_t` `mac_lld_get_transmit_descriptor` (`MACDriver` *macp, `MACTransmitDescriptor` *tdp)
Returns a transmission descriptor.
- void `mac_lld_release_transmit_descriptor` (`MACTransmitDescriptor` *tdp)

- **msg_t mac_lld_get_receive_descriptor (MACDriver *macp, MACReceiveDescriptor *rdp)**
Returns a receive descriptor.
- **void mac_lld_release_receive_descriptor (MACReceiveDescriptor *rdp)**
Releases a receive descriptor.
- **bool_t mac_lld_poll_link_status (MACDriver *macp)**
Updates and returns the link status.
- **size_t mac_lld_write_transmit_descriptor (MACTransmitDescriptor *tdp, uint8_t *buf, size_t size)**
Writes to a transmit descriptor's stream.
- **size_t mac_lld_read_receive_descriptor (MACReceiveDescriptor *rdp, uint8_t *buf, size_t size)**
Reads from a receive descriptor's stream.
- **uint8_t * mac_lld_get_next_transmit_buffer (MACTransmitDescriptor *tdp, size_t size, size_t *sizep)**
Returns a pointer to the next transmit buffer in the descriptor chain.
- **const uint8_t * mac_lld_get_next_receive_buffer (MACReceiveDescriptor *rdp, size_t *sizep)**
Returns a pointer to the next receive buffer in the descriptor chain.

Variables

- **MACDriver ETHD1**
MAC1 driver identifier.

13.88 mac_lld.h File Reference

13.88.1 Detailed Description

MAC Driver subsystem low level driver header template.

Data Structures

- struct **MACConfig**
Driver configuration structure.
- struct **MACDriver**
Structure representing a MAC driver.
- struct **MACTransmitDescriptor**
Structure representing a transmit descriptor.
- struct **MACReceiveDescriptor**
Structure representing a receive descriptor.

Functions

- **void mac_lld_init (void)**
Low level MAC initialization.
- **void mac_lld_start (MACDriver *macp)**
Configures and activates the MAC peripheral.
- **void mac_lld_stop (MACDriver *macp)**
Deactivates the MAC peripheral.
- **msg_t mac_lld_get_transmit_descriptor (MACDriver *macp, MACTransmitDescriptor *tdp)**
Returns a transmission descriptor.
- **void mac_lld_release_transmit_descriptor (MACTransmitDescriptor *tdp)**

Releases a transmit descriptor and starts the transmission of the enqueued data as a single frame.

- **msg_t mac_lld_get_receive_descriptor (MACDriver *macp, MACReceiveDescriptor *rdp)**
Returns a receive descriptor.
- **void mac_lld_release_receive_descriptor (MACReceiveDescriptor *rdp)**
Releases a receive descriptor.
- **bool_t mac_lld_poll_link_status (MACDriver *macp)**
Updates and returns the link status.
- **size_t mac_lld_write_transmit_descriptor (MACTransmitDescriptor *tdp, uint8_t *buf, size_t size)**
Writes to a transmit descriptor's stream.
- **size_t mac_lld_read_receive_descriptor (MACReceiveDescriptor *rdp, uint8_t *buf, size_t size)**
Reads from a receive descriptor's stream.

Defines

- **#define MAC_SUPPORTS_ZERO_COPY TRUE**
This implementation supports the zero-copy mode API.

Configuration options

- **#define PLATFORM_MAC_USE_MAC1 FALSE**
MAC driver enable switch.

13.89 memstreams.c File Reference

13.89.1 Detailed Description

```
Memory streams code. #include <string.h>
#include "ch.h"
#include "memstreams.h"
```

Functions

- **void msObjectInit (MemoryStream *msp, uint8_t *buffer, size_t size, size_t eos)**
Memory stream object initialization.

13.90 memstreams.h File Reference

13.90.1 Detailed Description

Memory streams structures and macros.

Data Structures

- **struct MemStreamVMT**
MemStream virtual methods table.
- **struct MemoryStream**
Memory stream object.

Functions

- void **msObjectInit** (**MemoryStream** *msp, **uint8_t** *buffer, **size_t** size, **size_t** eos)
Memory stream object initialization.

Defines

- **#define _memory_stream_data**
RamStream specific data.

13.91 mmc_spi.c File Reference

13.91.1 Detailed Description

MMC over SPI driver code.

```
#include <string.h>
#include "ch.h"
#include "hal.h"
```

Functions

- void **mmcInit** (void)
MMC over SPI driver initialization.
- void **mmcObjectInit** (**MMCDriver** *mmcp)
Initializes an instance.
- void **mmcStart** (**MMCDriver** *mmcp, const **MMCConfig** *config)
Configures and activates the MMC peripheral.
- void **mmcStop** (**MMCDriver** *mmcp)
Disables the MMC peripheral.
- **bool_t** **mmcConnect** (**MMCDriver** *mmcp)
Performs the initialization procedure on the inserted card.
- **bool_t** **mmcDisconnect** (**MMCDriver** *mmcp)
Brings the driver in a state safe for card removal.
- **bool_t** **mmcStartSequentialRead** (**MMCDriver** *mmcp, **uint32_t** startblk)
Starts a sequential read.
- **bool_t** **mmcSequentialRead** (**MMCDriver** *mmcp, **uint8_t** *buffer)
Reads a block within a sequential read operation.
- **bool_t** **mmcStopSequentialRead** (**MMCDriver** *mmcp)
Stops a sequential read gracefully.
- **bool_t** **mmcStartSequentialWrite** (**MMCDriver** *mmcp, **uint32_t** startblk)
Starts a sequential write.
- **bool_t** **mmcSequentialWrite** (**MMCDriver** *mmcp, const **uint8_t** *buffer)
Writes a block within a sequential write operation.
- **bool_t** **mmcStopSequentialWrite** (**MMCDriver** *mmcp)
Stops a sequential write gracefully.
- **bool_t** **mmcSync** (**MMCDriver** *mmcp)
Waits for card idle condition.
- **bool_t** **mmcGetInfo** (**MMCDriver** *mmcp, **BlockDeviceInfo** *bdip)
Returns the media info.
- **bool_t** **mmcErase** (**MMCDriver** *mmcp, **uint32_t** startblk, **uint32_t** endblk)
Erases blocks.

13.92 mmc_spi.h File Reference

13.92.1 Detailed Description

MMC over SPI driver header.

Data Structures

- struct [MMCConfig](#)
MMC/SD over SPI driver configuration structure.
- struct [MMCDriverVMT](#)
MMCDriver virtual methods table.
- struct [MMCDriver](#)
Structure representing a MMC/SD over SPI driver.

Functions

- void [mmcInit](#) (void)
MMC over SPI driver initialization.
- void [mmcObjectInit](#) ([MMCDriver](#) *mmcp)
Initializes an instance.
- void [mmcStart](#) ([MMCDriver](#) *mmcp, const [MMCConfig](#) *config)
Configures and activates the MMC peripheral.
- void [mmcStop](#) ([MMCDriver](#) *mmcp)
Disables the MMC peripheral.
- [bool_t](#) [mmcConnect](#) ([MMCDriver](#) *mmcp)
Performs the initialization procedure on the inserted card.
- [bool_t](#) [mmcDisconnect](#) ([MMCDriver](#) *mmcp)
Brings the driver in a state safe for card removal.
- [bool_t](#) [mmcStartSequentialRead](#) ([MMCDriver](#) *mmcp, [uint32_t](#) startblk)
Starts a sequential read.
- [bool_t](#) [mmcSequentialRead](#) ([MMCDriver](#) *mmcp, [uint8_t](#) *buffer)
Reads a block within a sequential read operation.
- [bool_t](#) [mmcStopSequentialRead](#) ([MMCDriver](#) *mmcp)
Stops a sequential read gracefully.
- [bool_t](#) [mmcStartSequentialWrite](#) ([MMCDriver](#) *mmcp, [uint32_t](#) startblk)
Starts a sequential write.
- [bool_t](#) [mmcSequentialWrite](#) ([MMCDriver](#) *mmcp, const [uint8_t](#) *buffer)
Writes a block within a sequential write operation.
- [bool_t](#) [mmcStopSequentialWrite](#) ([MMCDriver](#) *mmcp)
Stops a sequential write gracefully.
- [bool_t](#) [mmcSync](#) ([MMCDriver](#) *mmcp)
Waits for card idle condition.
- [bool_t](#) [mmcGetInfo](#) ([MMCDriver](#) *mmcp, [BlockDeviceInfo](#) *bdip)
Returns the media info.
- [bool_t](#) [mmcErase](#) ([MMCDriver](#) *mmcp, [uint32_t](#) startblk, [uint32_t](#) endblk)
Erases blocks.

Defines

- `#define _mmc_driver_methods _mmcsd_block_device_methods`
MMC Driver specific methods.

MMC_SPI configuration options

- `#define MMC_NICE_WAITING TRUE`
Delays insertions.

Macro Functions

- `#define mmclsCardInserted(mmcp) mmc_lld_is_card_inserted(mmcp)`
Returns the card insertion status.
- `#define mmclsWriteProtected(mmcp) mmc_lld_is_write_protected(mmcp)`
Returns the write protect status.

13.93 mmcsd.c File Reference

13.93.1 Detailed Description

MMC/SD cards common code.

```
#include "ch.h"
#include "hal.h"
```

Functions

- `uint32_t mmcsdGetCapacity (uint32_t csd[4])`
Extract card capacity from a CSD.

13.94 mmcsd.h File Reference

13.94.1 Detailed Description

MMC/SD cards common header. This header defines an abstract interface useful to access MMC/SD I/O block devices in a standardized way.

Data Structures

- struct `MMCSDBlockDeviceVMT`
MMCSDBlockDevice virtual methods table.
- struct `MMCSDBlockDevice`
MCC/SD block device class.

Functions

- `uint32_t mmcsdGetCapacity (uint32_t csd[4])`
Extract card capacity from a CSD.

Defines

- `#define MMCSD_BLOCK_SIZE 512`
Fixed block size for MMC/SD block devices.
- `#define MMCSD_R1_ERROR_MASK 0xFDFF0008`
Mask of error bits in R1 responses.
- `#define MMCSD_CMD8_PATTERN 0x000001AA`
Fixed pattern for CMD8.
- `#define _mmcisd_block_device_methods _base_block_device_methods`
MMCSDBlockDevice specific methods.
- `#define _mmcisd_block_device_data`
MMCSDBlockDevice specific data.

SD/MMC status conditions

- `#define MMCSD_STS_IDLE 0`
- `#define MMCSD_STS_READY 1`
- `#define MMCSD_STS_IDENT 2`
- `#define MMCSD_STS_STBY 3`
- `#define MMCSD_STS_TRAN 4`
- `#define MMCSD_STS_DATA 5`
- `#define MMCSD_STS_RCV 6`
- `#define MMCSD_STS_PRG 7`
- `#define MMCSD_STS_DIS 8`

SD/MMC commands

- `#define MMCSD_CMD_GO_IDLE_STATE 0`
- `#define MMCSD_CMD_INIT 1`
- `#define MMCSD_CMD_ALL_SEND_CID 2`
- `#define MMCSD_CMD_SEND_RELATIVE_ADDR 3`
- `#define MMCSD_CMD_SET_BUS_WIDTH 6`
- `#define MMCSD_CMD_SEL_DESEL_CARD 7`
- `#define MMCSD_CMD_SEND_IF_COND 8`
- `#define MMCSD_CMD_SEND_CSD 9`
- `#define MMCSD_CMD_SEND_CID 10`
- `#define MMCSD_CMD_STOP_TRANSMISSION 12`
- `#define MMCSD_CMD_SEND_STATUS 13`
- `#define MMCSD_CMD_SET_BLOCKLEN 16`
- `#define MMCSD_CMD_READ_SINGLE_BLOCK 17`
- `#define MMCSD_CMD_READ_MULTIPLE_BLOCK 18`
- `#define MMCSD_CMD_SET_BLOCK_COUNT 23`
- `#define MMCSD_CMD_WRITE_BLOCK 24`
- `#define MMCSD_CMD_WRITE_MULTIPLE_BLOCK 25`
- `#define MMCSD_CMD_ERASE_RW_BLK_START 32`
- `#define MMCSD_CMD_ERASE_RW_BLK_END 33`
- `#define MMCSD_CMD_ERASE 38`
- `#define MMCSD_CMD_APP_OP_COND 41`
- `#define MMCSD_CMD_LOCK_UNLOCK 42`
- `#define MMCSD_CMD_APP_CMD 55`
- `#define MMCSD_CMD_READ_OCR 58`

CSD record offsets

- `#define MMCSD_CSD_20_CRC_SLICE 7,1`
Slice position of values in CSD register.
- `#define MMCSD_CSD_20_FILE_FORMAT_SLICE 11,10`
- `#define MMCSD_CSD_20_TMP_WRITE_PROTECT_SLICE 12,12`
- `#define MMCSD_CSD_20_PERM_WRITE_PROTECT_SLICE 13,13`
- `#define MMCSD_CSD_20_COPY_SLICE 14,14`
- `#define MMCSD_CSD_20_FILE_FORMAT_GRP_SLICE 15,15`

- #define **MMCSD_CSD_20_WRITE_BL_PARTIAL_SLICE** 21,21
- #define **MMCSD_CSD_20_WRITE_BL_LEN_SLICE** 25,12
- #define **MMCSD_CSD_20_R2W_FACTOR_SLICE** 28,26
- #define **MMCSD_CSD_20_WP_GRP_ENABLE_SLICE** 31,31
- #define **MMCSD_CSD_20_WP_GRP_SIZE_SLICE** 38,32
- #define **MMCSD_CSD_20_ERASE_SECTOR_SIZE_SLICE** 45,39
- #define **MMCSD_CSD_20_ERASE_BLK_EN_SLICE** 46,46
- #define **MMCSD_CSD_20_C_SIZE_SLICE** 69,48
- #define **MMCSD_CSD_20_DSR_IMP_SLICE** 76,76
- #define **MMCSD_CSD_20_READ_BLK_MISALIGN_SLICE** 77,77
- #define **MMCSD_CSD_20_WRITE_BLK_MISALIGN_SLICE** 78,78
- #define **MMCSD_CSD_20_READ_BL_PARTIAL_SLICE** 79,79
- #define **MMCSD_CSD_20_READ_BL_LEN_SLICE** 83,80
- #define **MMCSD_CSD_20_CCC_SLICE** 95,84
- #define **MMCSD_CSD_20_TRANS_SPEED_SLICE** 103,96
- #define **MMCSD_CSD_20_NSAC_SLICE** 111,104
- #define **MMCSD_CSD_20_TAAC_SLICE** 119,112
- #define **MMCSD_CSD_20_STRUCTURE_SLICE** 127,126
- #define **MMCSD_CSD_10_CRC_SLICE** MMCSD_CSD_20_CRC_SLICE
- #define **MMCSD_CSD_10_FILE_FORMAT_SLICE** MMCSD_CSD_20_FILE_FORMAT_SLICE
- #define **MMCSD_CSD_10_TMP_WRITE_PROTECT_SLICE** MMCSD_CSD_20_TMP_WRITE_PROTECT_SLICE
- #define **MMCSD_CSD_10_PERM_WRITE_PROTECT_SLICE** MMCSD_CSD_20_PERM_WRITE_PROTECT_SLICE
- #define **MMCSD_CSD_10_COPY_SLICE** MMCSD_CSD_20_COPY_SLICE
- #define **MMCSD_CSD_10_FILE_FORMAT_GRP_SLICE** MMCSD_CSD_20_FILE_FORMAT_GRP_SLICE
- #define **MMCSD_CSD_10_WRITE_BL_PARTIAL_SLICE** MMCSD_CSD_20_WRITE_BL_PARTIAL_SLICE
- #define **MMCSD_CSD_10_WRITE_BL_LEN_SLICE** MMCSD_CSD_20_WRITE_BL_LEN_SLICE
- #define **MMCSD_CSD_10_R2W_FACTOR_SLICE** MMCSD_CSD_20_R2W_FACTOR_SLICE
- #define **MMCSD_CSD_10_WP_GRP_ENABLE_SLICE** MMCSD_CSD_20_WP_GRP_ENABLE_SLICE
- #define **MMCSD_CSD_10_WP_GRP_SIZE_SLICE** MMCSD_CSD_20_WP_GRP_SIZE_SLICE
- #define **MMCSD_CSD_10_ERASE_SECTOR_SIZE_SLICE** MMCSD_CSD_20_ERASE_SECTOR_SIZE_SLICE
- #define **MMCSD_CSD_10_ERASE_BLK_EN_SLICE** MMCSD_CSD_20_ERASE_BLK_EN_SLICE
- #define **MMCSD_CSD_10_C_SIZE_MULT_SLICE** 49,47
- #define **MMCSD_CSD_10_VDD_W_CURR_MAX_SLICE** 52,50
- #define **MMCSD_CSD_10_VDD_W_CURR_MIN_SLICE** 55,53
- #define **MMCSD_CSD_10_VDD_R_CURR_MAX_SLICE** 58,56
- #define **MMCSD_CSD_10_VDD_R_CURR_MIX_SLICE** 61,59
- #define **MMCSD_CSD_10_C_SIZE_SLICE** 73,62
- #define **MMCSD_CSD_10_DSR_IMP_SLICE** MMCSD_CSD_20_DSR_IMP_SLICE
- #define **MMCSD_CSD_10_READ_BLK_MISALIGN_SLICE** MMCSD_CSD_20_READ_BLK_MISALIGN_SLICE
- #define **MMCSD_CSD_10_WRITE_BLK_MISALIGN_SLICE** MMCSD_CSD_20_WRITE_BLK_MISALIGN_SLICE
- #define **MMCSD_CSD_10_READ_BL_PARTIAL_SLICE** MMCSD_CSD_20_READ_BL_PARTIAL_SLICE
- #define **MMCSD_CSD_10_READ_BL_LEN_SLICE** 83,80
- #define **MMCSD_CSD_10_CCC_SLICE** MMCSD_CSD_20_CCC_SLICE
- #define **MMCSD_CSD_10_TRANS_SPEED_SLICE** MMCSD_CSD_20_TRANS_SPEED_SLICE
- #define **MMCSD_CSD_10_NSAC_SLICE** MMCSD_CSD_20_NSAC_SLICE
- #define **MMCSD_CSD_10_TAAC_SLICE** MMCSD_CSD_20_TAAC_SLICE
- #define **MMCSD_CSD_10_STRUCTURE_SLICE** MMCSD_CSD_20_STRUCTURE_SLICE

R1 response utilities

- #define **MMCSD_R1_ERROR**(r1) (((r1) & MMCSD_R1_ERROR_MASK) != 0)

Evaluates to TRUE if the R1 response contains error flags.
- #define **MMCSD_R1_STS**(r1) (((r1) >> 9) & 15)

Returns the status field of an R1 response.
- #define **MMCSD_R1_IS_CARD_LOCKED**(r1) (((r1) >> 21) & 1)

Evaluates to TRUE if the R1 response indicates a locked card.

Macro Functions

- `#define mmcsdGetCardCapacity(ip) ((ip)->capacity)`
Returns the card capacity in blocks.

13.95 pal.c File Reference

13.95.1 Detailed Description

I/O Ports Abstraction Layer code.

```
#include "ch.h"
#include "hal.h"
```

Functions

- `ioportmask_t palReadBus (IOBus *bus)`
Read from an I/O bus.
- `void palWriteBus (IOBus *bus, ioportmask_t bits)`
Write to an I/O bus.
- `void palSetBusMode (IOBus *bus, iomode_t mode)`
Programs a bus with the specified mode.

13.96 pal.h File Reference

13.96.1 Detailed Description

I/O Ports Abstraction Layer macros, types and structures.

```
#include "pal_lld.h"
```

Data Structures

- `struct IOBus`
I/O bus descriptor.

Functions

- `ioportmask_t palReadBus (IOBus *bus)`
Read from an I/O bus.
- `void palWriteBus (IOBus *bus, ioportmask_t bits)`
Write to an I/O bus.
- `void palSetBusMode (IOBus *bus, iomode_t mode)`
Programs a bus with the specified mode.

Defines

- `#define PAL_PORT_BIT(n) ((ioportmask_t)(1 << (n)))`
Port bit helper macro.
- `#define PAL_GROUP_MASK(width) ((ioportmask_t)(1 << (width)) - 1)`
Bits group mask helper.
- `#define _IOBUS_DATA(name, port, width, offset) {port, PAL_GROUP_MASK(width), offset}`
Data part of a static I/O bus initializer.

- `#define IOBUS_DECL(name, port, width, offset) IOBus name = _IOBUS_DATA(name, port, width, offset)`
Static I/O bus initializer.

Pads mode constants

- `#define PAL_MODE_RESET 0`
After reset state.
- `#define PAL_MODE_UNCONNECTED 1`
*Safe state for **unconnected** pads.*
- `#define PAL_MODE_INPUT 2`
Regular input high-Z pad.
- `#define PAL_MODE_INPUT_PULLUP 3`
Input pad with weak pull up resistor.
- `#define PAL_MODE_INPUT_PULLDOWN 4`
Input pad with weak pull down resistor.
- `#define PAL_MODE_INPUT_ANALOG 5`
Analog input mode.
- `#define PAL_MODE_OUTPUT_PUSH_PULL 6`
Push-pull output pad.
- `#define PAL_MODE_OUTPUT_OPENDRAIN 7`
Open-drain output pad.

Logic level constants

- `#define PAL_LOW 0`
Logical low state.
- `#define PAL_HIGH 1`
Logical high state.

Macro Functions

- `#define palInit(config) pal_lld_init(config)`
PAL subsystem initialization.
- `#define palReadPort(port) ((void)(port), 0)`
Reads the physical I/O port states.
- `#define palReadLatch(port) ((void)(port), 0)`
Reads the output latch.
- `#define palWritePort(port, bits) ((void)(port), (void)(bits))`
Writes a bits mask on a I/O port.
- `#define palSetPort(port, bits) palWritePort(port, palReadLatch(port) | (bits))`
Sets a bits mask on a I/O port.
- `#define palClearPort(port, bits) palWritePort(port, palReadLatch(port) & ~ (bits))`
Clears a bits mask on a I/O port.
- `#define palTogglePort(port, bits) palWritePort(port, palReadLatch(port) ^ (bits))`
Toggles a bits mask on a I/O port.
- `#define palReadGroup(port, mask, offset) ((palReadPort(port) >> (offset)) & (mask))`
Reads a group of bits.
- `#define palWriteGroup(port, mask, offset, bits)`
Writes a group of bits.
- `#define palSetGroupMode(port, mask, offset, mode)`
Pads group mode setup.
- `#define palReadPad(port, pad) ((palReadPort(port) >> (pad)) & 1)`
Reads an input pad logical state.
- `#define palWritePad(port, pad, bit)`
Writes a logical state on an output pad.
- `#define palSetPad(port, pad) palSetPort(port, PAL_PORT_BIT(pad))`
*Sets a pad logical state to **PAL_HIGH**.*
- `#define palClearPad(port, pad) palClearPort(port, PAL_PORT_BIT(pad))`
*Clears a pad logical state to **PAL_LOW**.*
- `#define palTogglePad(port, pad) palTogglePort(port, PAL_PORT_BIT(pad))`
Toggles a pad logical state.
- `#define palSetPadMode(port, pad, mode) palSetGroupMode(port, PAL_PORT_BIT(pad), 0, mode)`
Pad mode setup.

13.97 pal_lld.c File Reference

13.97.1 Detailed Description

PAL subsystem low level driver template.

```
#include "ch.h"
#include "hal.h"
```

Functions

- void `_pal_lld_init` (const `PALConfig` *config)
STM32 I/O ports configuration.
- void `_pal_lld_setgroupmode` (`ioportid_t` port, `ioportmask_t` mask, `iomode_t` mode)
Pads mode setup.

13.98 pal_lld.h File Reference

13.98.1 Detailed Description

PAL subsystem low level driver header template.

Data Structures

- struct `PALConfig`
Generic I/O ports static initializer.

Functions

- void `_pal_lld_init` (const `PALConfig` *config)
STM32 I/O ports configuration.
- void `_pal_lld_setgroupmode` (`ioportid_t` port, `ioportmask_t` mask, `iomode_t` mode)
Pads mode setup.

Defines

- `#define PAL_IOPORTS_WIDTH 32`
Width, in bits, of an I/O port.
- `#define PAL_WHOLE_PORT ((ioportmask_t)0xFFFFFFFF)`
Whole port mask.
- `#define IOPORT1 0`
First I/O port identifier.
- `#define pal_lld_init(config) _pal_lld_init(config)`
Low level PAL subsystem initialization.
- `#define pal_lld_readport(port) 0`
Reads the physical I/O port states.
- `#define pal_lld_readlatch(port) 0`
Reads the output latch.
- `#define pal_lld_writeport(port, bits)`
Writes a bits mask on a I/O port.
- `#define pal_lld_setport(port, bits)`

- **#define pal_lld_clearport(port, bits)**
Sets a bits mask on a I/O port.
- **#define pal_lld_toggleport(port, bits)**
Clears a bits mask on a I/O port.
- **#define pal_lld_readgroup(port, mask, offset) 0**
Toggles a bits mask on a I/O port.
- **#define pal_lld_readgroup(port, mask, offset, bits)**
Reads a group of bits.
- **#define pal_lld_writegroup(port, mask, offset, bits) (void)bits**
Writes a group of bits.
- **#define pal_lld_setgroupmode(port, mask, offset, mode) _pal_lld_setgroupmode(port, mask << offset, mode)**
Pads group mode setup.
- **#define pal_lld_readpad(port, pad) PAL_LOW**
Reads a logical state from an I/O pad.
- **#define pal_lld_writepad(port, pad, bit)**
Writes a logical state on an output pad.
- **#define pal_lld_setpad(port, pad)**
Sets a pad logical state to PAL_HIGH.
- **#define pal_lld_clearpad(port, pad)**
Clears a pad logical state to PAL_LOW.
- **#define pal_lld_togglepad(port, pad)**
Toggles a pad logical state.
- **#define pal_lld_setpadmode(port, pad, mode)**
Pad mode setup.

Typedefs

- **typedef uint32_t ioportmask_t**
Digital I/O port sized unsigned type.
- **typedef uint32_t iomode_t**
Digital I/O modes.
- **typedef uint32_t ioportid_t**
Port Identifier.

13.99 pwm.c File Reference

13.99.1 Detailed Description

```
PWM Driver code. #include "ch.h"
#include "hal.h"
```

Functions

- **void pwmlInit (void)**
PWM Driver initialization.
- **void pwmObjectInit (PWMDriver *pwmp)**
Initializes the standard part of a PWMDriver structure.
- **void pwmStart (PWMDriver *pwmp, const PWMConfig *config)**
Configures and activates the PWM peripheral.

- void **pwmStop** (**PWMDriver** *pwmp)
Deactivates the PWM peripheral.
- void **pwmChangePeriod** (**PWMDriver** *pwmp, **pwmcnt_t** period)
Changes the period the PWM peripheral.
- void **pwmEnableChannel** (**PWMDriver** *pwmp, **pwmchannel_t** channel, **pwmcnt_t** width)
Enables a PWM channel.
- void **pwmDisableChannel** (**PWMDriver** *pwmp, **pwmchannel_t** channel)
Disables a PWM channel.

13.100 pwm.h File Reference

13.100.1 Detailed Description

PWM Driver macros and structures. #include "pwm_lld.h"

Functions

- void **pwmInit** (void)
PWM Driver initialization.
- void **pwmObjectInit** (**PWMDriver** *pwmp)
*Initializes the standard part of a **PWMDriver** structure.*
- void **pwmStart** (**PWMDriver** *pwmp, const **PWMConfig** *config)
Configures and activates the PWM peripheral.
- void **pwmStop** (**PWMDriver** *pwmp)
Deactivates the PWM peripheral.
- void **pwmChangePeriod** (**PWMDriver** *pwmp, **pwmcnt_t** period)
Changes the period the PWM peripheral.
- void **pwmEnableChannel** (**PWMDriver** *pwmp, **pwmchannel_t** channel, **pwmcnt_t** width)
Enables a PWM channel.
- void **pwmDisableChannel** (**PWMDriver** *pwmp, **pwmchannel_t** channel)
Disables a PWM channel.

Defines

PWM output mode macros

- #define **PWM_OUTPUT_MASK** 0x0F
Standard output modes mask.
- #define **PWM_OUTPUT_DISABLED** 0x00
Output not driven, callback only.
- #define **PWM_OUTPUT_ACTIVE_HIGH** 0x01
Positive PWM logic, active is logic level one.
- #define **PWM_OUTPUT_ACTIVE_LOW** 0x02
Inverse PWM logic, active is logic level zero.

PWM duty cycle conversion

- #define **PWM_FRACTION_TO_WIDTH**(pwmp, denominator, numerator)
Converts from fraction to pulse width.
- #define **PWM_DEGREES_TO_WIDTH**(pwmp, degrees) **PWM_FRACTION_TO_WIDTH**(pwmp, 36000, degrees)
Converts from degrees to pulse width.
- #define **PWM_PERCENTAGE_TO_WIDTH**(pwmp, percentage) **PWM_FRACTION_TO_WIDTH**(pwmp, 10000, percentage)
Converts from percentage to pulse width.

Macro Functions

- `#define pwmChangePeriodI(pwmp, value)`
Changes the period the PWM peripheral.
- `#define pwmEnableChannelI(pwmp, channel, width) pwm_lld_enable_channel(pwmp, channel, width)`
Enables a PWM channel.
- `#define pwmDisableChannelI(pwmp, channel) pwm_lld_disable_channel(pwmp, channel)`
Disables a PWM channel.
- `#define pwmIsChannelEnabledI(pwmp, channel) pwm_lld_is_channel_enabled(pwmp, channel)`
Returns a PWM channel status.

TypeDefs

- `typedef struct PWMDriver PWMDriver`
Type of a structure representing a PWM driver.
- `typedef void(* pwmcallback_t)(PWMDriver *pwmp)`
PWM notification callback type.

Enumerations

- `enum pwmstate_t { PWM_UNINIT = 0, PWM_STOP = 1, PWM_READY = 2 }`
Driver state machine possible states.

13.101 pwm_lld.c File Reference

13.101.1 Detailed Description

PWM Driver subsystem low level driver source template.

```
#include "ch.h"
#include "hal.h"
```

Functions

- `void pwm_lld_init (void)`
Low level PWM driver initialization.
- `void pwm_lld_start (PWMDriver *pwmp)`
Configures and activates the PWM peripheral.
- `void pwm_lld_stop (PWMDriver *pwmp)`
Deactivates the PWM peripheral.
- `void pwm_lld_change_period (PWMDriver *pwmp, pwcnt_t period)`
Changes the period the PWM peripheral.
- `void pwm_lld_enable_channel (PWMDriver *pwmp, pwmchannel_t channel, pwcnt_t width)`
Enables a PWM channel.
- `void pwm_lld_disable_channel (PWMDriver *pwmp, pwmchannel_t channel)`
Disables a PWM channel.

Variables

- `PWMDriver PWMD1`
PWM1 driver identifier.

13.102 pwm_lld.h File Reference

13.102.1 Detailed Description

PWM Driver subsystem low level driver header template.

Data Structures

- struct [PWMChannelConfig](#)
PWM driver channel configuration structure.
- struct [PWMConfig](#)
Driver configuration structure.
- struct [PWMDriver](#)
Structure representing an PWM driver.

Functions

- void [pwm_lld_init](#) (void)
Low level PWM driver initialization.
- void [pwm_lld_start](#) ([PWMDriver](#) *pwmp)
Configures and activates the PWM peripheral.
- void [pwm_lld_stop](#) ([PWMDriver](#) *pwmp)
Deactivates the PWM peripheral.
- void [pwm_lld_change_period](#) ([PWMDriver](#) *pwmp, [pwmcnt_t](#) period)
Changes the period the PWM peripheral.
- void [pwm_lld_enable_channel](#) ([PWMDriver](#) *pwmp, [pwmchannel_t](#) channel, [pwmcnt_t](#) width)
Enables a PWM channel.
- void [pwm_lld_disable_channel](#) ([PWMDriver](#) *pwmp, [pwmchannel_t](#) channel)
Disables a PWM channel.

Defines

- #define [PWM_CHANNELS](#) 4
Number of PWM channels per PWM driver.
- #define [pwm_lld_is_channel_enabled](#)(pwmp, channel) FALSE
Returns a PWM channel status.

Configuration options

- #define [PLATFORM_PWM_USE_PWM1](#) FALSE
PWM driver enable switch.

Typedefs

- typedef uint32_t [pwmmodet](#)
PWM mode type.
- typedef uint8_t [pwmchannel_t](#)
PWM channel type.
- typedef uint16_t [pwmcnt_t](#)
PWM counter type.

13.103 rtc.c File Reference

13.103.1 Detailed Description

RTC Driver code.

```
#include "ch.h"
#include "hal.h"
```

Functions

- void `rtcInit` (void)
RTC Driver initialization.
- void `rtcSetTime` (RTCDriver *rtcp, const RTCTime *timespec)
Set current time.
- void `rtcGetTime` (RTCDriver *rtcp, RTCTime *timespec)
Get current time.
- void `rtcSetAlarm` (RTCDriver *rtcp, rtcalarm_t alarm, const RTCAlarm *alarmspec)
Set alarm time.
- void `rtcGetAlarm` (RTCDriver *rtcp, rtcalarm_t alarm, RTCAlarm *alarmspec)
Get current alarm.
- void `rtcSetCallback` (RTCDriver *rtcp, rtccb_t callback)
Enables or disables RTC callbacks.
- uint32_t `rtcGetTimeFat` (RTCDriver *rtcp)
Get current time in format suitable for usage in FatFS.

13.104 rtc.h File Reference

13.104.1 Detailed Description

RTC Driver macros and structures.

```
#include "rtc_lld.h"
```

Functions

- void `rtcInit` (void)
RTC Driver initialization.
- void `rtcSetTime` (RTCDriver *rtcp, const RTCTime *timespec)
Set current time.
- void `rtcGetTime` (RTCDriver *rtcp, RTCTime *timespec)
Get current time.
- uint32_t `rtcGetTimeFat` (RTCDriver *rtcp)
Get current time in format suitable for usage in FatFS.

Defines

- #define `rtcSetTimel`(rtcp, timespec) `rtc_lld_set_time`(rtcp, timespec)
Set current time.
- #define `rtcGetTimel`(rtcp, timespec) `rtc_lld_get_time`(rtcp, timespec)
Get current time.
- #define `rtcSetAlarms`(rtcp, alarm, alarmspec) `rtc_lld_set_alarm`(rtcp, alarm, alarmspec)
Set alarm time.

- `#define rtcGetAlarmI(rtcp, alarm, alarmspec) rtc_lld_get_alarm(rtcp, alarm, alarmspec)`
Get current alarm.
- `#define rtcSetCallbackI(rtcp, callback) rtc_lld_set_callback(rtcp, callback)`
Enables or disables RTC callbacks.

Date/Time bit masks

- `#define RTC_TIME_SECONDS_MASK 0x00000001F`
- `#define RTC_TIME_MINUTES_MASK 0x0000007E0`
- `#define RTC_TIME_HOURS_MASK 0x0000F800`
- `#define RTC_DATE_DAYS_MASK 0x001F0000`
- `#define RTC_DATE_MONTHS_MASK 0x01E00000`
- `#define RTC_DATE_YEARS_MASK 0xFE000000`

Typedefs

- `typedef struct RTCDriver RTCDriver`
Type of a structure representing an RTC driver.
- `typedef struct RTCTime RTCTime`
Type of a structure representing an RTC time stamp.

13.105 sdc.c File Reference

13.105.1 Detailed Description

SDC Driver code.

```
#include "ch.h"
#include "hal.h"
```

Functions

- `bool_t _sdc_wait_for_transfer_state (SDCDriver *sdcp)`
Wait for the card to complete pending operations.
- `void sdclinit (void)`
SDC Driver initialization.
- `void sdcObjectInit (SDCDriver *sdcp)`
Initializes the standard part of a `SDCDriver` structure.
- `void sdcStart (SDCDriver *sdcp, const SDCCConfig *config)`
Configures and activates the SDC peripheral.
- `void sdcStop (SDCDriver *sdcp)`
Deactivates the SDC peripheral.
- `bool_t sdcConnect (SDCDriver *sdcp)`
Performs the initialization procedure on the inserted card.
- `bool_t sdcDisconnect (SDCDriver *sdcp)`
Brings the driver in a state safe for card removal.
- `bool_t sdcRead (SDCDriver *sdcp, uint32_t startblk, uint8_t *buf, uint32_t n)`
Reads one or more blocks.
- `bool_t sdcWrite (SDCDriver *sdcp, uint32_t startblk, const uint8_t *buf, uint32_t n)`
Writes one or more blocks.
- `sdcflags_t sdcGetAndClearErrors (SDCDriver *sdcp)`
Returns the errors mask associated to the previous operation.
- `bool_t sdcSync (SDCDriver *sdcp)`

- **bool_t sdcGetInfo (SDCDriver *sdcp, BlockDeviceInfo *bdip)**
Returns the media info.
- **bool_t sdcErase (SDCDriver *sdcp, uint32_t startblk, uint32_t endblk)**
Erases the supplied blocks.

13.106 sdc.h File Reference

13.106.1 Detailed Description

SDC Driver macros and structures. #include "sdc_lld.h"

Functions

- **void sdclnit (void)**
SDC Driver initialization.
- **void sdcObjectInit (SDCDriver *sdcp)**
Initializes the standard part of a `SDCDriver` structure.
- **void sdcStart (SDCDriver *sdcp, const SDCCConfig *config)**
Configures and activates the SDC peripheral.
- **void sdcStop (SDCDriver *sdcp)**
Deactivates the SDC peripheral.
- **bool_t sdcConnect (SDCDriver *sdcp)**
Performs the initialization procedure on the inserted card.
- **bool_t sdcDisconnect (SDCDriver *sdcp)**
Brings the driver in a state safe for card removal.
- **bool_t sdcRead (SDCDriver *sdcp, uint32_t startblk, uint8_t *buf, uint32_t n)**
Reads one or more blocks.
- **bool_t sdcWrite (SDCDriver *sdcp, uint32_t startblk, const uint8_t *buf, uint32_t n)**
Writes one or more blocks.
- **sdcflags_t sdcGetAndClearErrors (SDCDriver *sdcp)**
Returns the errors mask associated to the previous operation.
- **bool_t sdcSync (SDCDriver *sdcp)**
Waits for card idle condition.
- **bool_t sdcGetInfo (SDCDriver *sdcp, BlockDeviceInfo *bdip)**
Returns the media info.
- **bool_t sdcErase (SDCDriver *sdcp, uint32_t startblk, uint32_t endblk)**
Erases the supplied blocks.
- **bool_t _sdc_wait_for_transfer_state (SDCDriver *sdcp)**
Wait for the card to complete pending operations.

Defines

SD cart types

- **#define SDC_MODE_CARDTYPE_MASK 0xF**
Card type mask.
- **#define SDC_MODE_CARDTYPE_SDV11 0**
Card is SD V1.1.
- **#define SDC_MODE_CARDTYPE_SDV20 1**
Card is SD V2.0.

- #define **SDC_MODE_CARDTYPE_MMC** 2
Card is MMC.
- #define **SDC_MODE_HIGH_CAPACITY** 0x10
High cap.card.

SDC bus error conditions

- #define **SDC_NO_ERROR** 0
No error.
- #define **SDC_CMD_CRC_ERROR** 1
Command CRC error.
- #define **SDC_DATA_CRC_ERROR** 2
Data CRC error.
- #define **SDC_DATA_TIMEOUT** 4
HW write timeout.
- #define **SDC_COMMAND_TIMEOUT** 8
HW read timeout.
- #define **SDC_TX_UNDERRUN** 16
TX buffer underrun.
- #define **SDC_RX_OVERRUN** 32
RX buffer overrun.
- #define **SDC_STARTBIT_ERROR** 64
Start bit missing.
- #define **SDC_OVERFLOW_ERROR** 128
Card overflow error.
- #define **SDC_UNHANDLED_ERROR** 0xFFFFFFFF

SDC configuration options

- #define **SDC_INIT_RETRY** 100
Number of initialization attempts before rejecting the card.
- #define **SDC_MMIC_SUPPORT** FALSE
Include support for MMC cards.
- #define **SDC_NICE_WAITING** TRUE
Delays insertions.

Macro Functions

- #define **sdclsCardInserted**(sdcp) (**sdc_lld_is_card_inserted**(sdcp))
Returns the card insertion status.
- #define **sdclsWriteProtected**(sdcp) (**sdc_lld_is_write_protected**(sdcp))
Returns the write protect status.

13.107 sdc_lld.c File Reference

13.107.1 Detailed Description

SDC Driver subsystem low level driver source template.

```
#include "ch.h"
#include "hal.h"
```

Functions

- void **sdc_lld_init** (void)
Low level SDC driver initialization.
- void **sdc_lld_start** (**SDCDriver** *sdcp)
Configures and activates the SDC peripheral.

- void `sdc_lld_stop (SDCDriver *sdcp)`
Deactivates the SDC peripheral.
- void `sdc_lld_start_clk (SDCDriver *sdcp)`
Starts the SDIO clock and sets it to init mode (400kHz or less).
- void `sdc_lld_set_data_clk (SDCDriver *sdcp)`
Sets the SDIO clock to data mode (25MHz or less).
- void `sdc_lld_stop_clk (SDCDriver *sdcp)`
Stops the SDIO clock.
- void `sdc_lld_set_bus_mode (SDCDriver *sdcp, sdcbusmode_t mode)`
Switches the bus to 4 bits mode.
- void `sdc_lld_send_cmd_none (SDCDriver *sdcp, uint8_t cmd, uint32_t arg)`
Sends an SDIO command with no response expected.
- `bool_t sdc_lld_send_cmd_short (SDCDriver *sdcp, uint8_t cmd, uint32_t arg, uint32_t *resp)`
Sends an SDIO command with a short response expected.
- `bool_t sdc_lld_send_cmd_short_crc (SDCDriver *sdcp, uint8_t cmd, uint32_t arg, uint32_t *resp)`
Sends an SDIO command with a short response expected and CRC.
- `bool_t sdc_lld_send_cmd_long_crc (SDCDriver *sdcp, uint8_t cmd, uint32_t arg, uint32_t *resp)`
Sends an SDIO command with a long response expected and CRC.
- `bool_t sdc_lld_read (SDCDriver *sdcp, uint32_t startblk, uint8_t *buf, uint32_t n)`
Reads one or more blocks.
- `bool_t sdc_lld_write (SDCDriver *sdcp, uint32_t startblk, const uint8_t *buf, uint32_t n)`
Writes one or more blocks.
- `bool_t sdc_lld_sync (SDCDriver *sdcp)`
Waits for card idle condition.

Variables

- `SDCDriver SDCD1`
SDCD1 driver identifier.

13.108 sdc_lld.h File Reference

13.108.1 Detailed Description

SDC Driver subsystem low level driver header template.

Data Structures

- struct `SDCConfig`
Driver configuration structure.
- struct `SDCDriverVMT`
SDCDriver virtual methods table.
- struct `SDCDriver`
Structure representing an SDC driver.

Functions

- void `sdc_ll_init` (void)
Low level SDC driver initialization.
- void `sdc_ll_start` (`SDCDriver` *sdcp)
Configures and activates the SDC peripheral.
- void `sdc_ll_stop` (`SDCDriver` *sdcp)
Deactivates the SDC peripheral.
- void `sdc_ll_start_clk` (`SDCDriver` *sdcp)
Starts the SDIO clock and sets it to init mode (400kHz or less).
- void `sdc_ll_set_data_clk` (`SDCDriver` *sdcp)
Sets the SDIO clock to data mode (25MHz or less).
- void `sdc_ll_stop_clk` (`SDCDriver` *sdcp)
Stops the SDIO clock.
- void `sdc_ll_set_bus_mode` (`SDCDriver` *sdcp, `sdcbusmode_t` mode)
Switches the bus to 4 bits mode.
- void `sdc_ll_send_cmd_none` (`SDCDriver` *sdcp, `uint8_t` cmd, `uint32_t` arg)
Sends an SDIO command with no response expected.
- `bool_t sdc_ll_send_cmd_short` (`SDCDriver` *sdcp, `uint8_t` cmd, `uint32_t` arg, `uint32_t` *resp)
Sends an SDIO command with a short response expected.
- `bool_t sdc_ll_send_cmd_short_crc` (`SDCDriver` *sdcp, `uint8_t` cmd, `uint32_t` arg, `uint32_t` *resp)
Sends an SDIO command with a short response expected and CRC.
- `bool_t sdc_ll_send_cmd_long_crc` (`SDCDriver` *sdcp, `uint8_t` cmd, `uint32_t` arg, `uint32_t` *resp)
Sends an SDIO command with a long response expected and CRC.
- `bool_t sdc_ll_read` (`SDCDriver` *sdcp, `uint32_t` startblk, `uint8_t` *buf, `uint32_t` n)
Reads one or more blocks.
- `bool_t sdc_ll_write` (`SDCDriver` *sdcp, `uint32_t` startblk, `const uint8_t` *buf, `uint32_t` n)
Writes one or more blocks.
- `bool_t sdc_ll_sync` (`SDCDriver` *sdcp)
Waits for card idle condition.

Defines

- `#define _sdc_driver_methods _mmcsd_block_device_methods`
SDC driver specific methods.

Configuration options

- `#define PLATFORM_SDC_USE_SDC1 TRUE`
SDC driver enable switch.

R1 response utilities

- `#define MMCSD_R1_ERROR(r1) (((r1) & MMCSD_R1_ERROR_MASK) != 0)`
Evaluates to TRUE if the R1 response contains error flags.
- `#define MMCSD_R1_STS(r1) (((r1) >> 9) & 15)`
Returns the status field of an R1 response.
- `#define MMCSD_R1_IS_CARD_LOCKED(r1) (((r1) >> 21) & 1)`
Evaluates to TRUE if the R1 response indicates a locked card.

Typedefs

- `typedef uint32_t sdcmode_t`
Type of card flags.
- `typedef uint32_t sdcflags_t`
SDC Driver condition flags type.
- `typedef struct SDCDriver SDCDriver`
Type of a structure representing an SDC driver.

Enumerations

- `enum sdcbusmode_t`
Type of SDIO bus mode.

13.109 serial.c File Reference

13.109.1 Detailed Description

Serial Driver code.

```
#include "ch.h"
#include "hal.h"
```

Functions

- `void sdInit (void)`
Serial Driver initialization.
- `void sdObjectInit (SerialDriver *sdp, qnotify_t inotify, qnotify_t onotify)`
Initializes a generic full duplex driver object.
- `void sdStart (SerialDriver *sdp, const SerialConfig *config)`
Configures and starts the driver.
- `void sdStop (SerialDriver *sdp)`
Stops the driver.
- `void sdIncomingData (SerialDriver *sdp, uint8_t b)`
Handles incoming data.
- `msg_t sdRequestData (SerialDriver *sdp)`
Handles outgoing data.

13.110 serial.h File Reference

13.110.1 Detailed Description

Serial Driver macros and structures.

```
#include "serial_lld.h"
```

Data Structures

- `struct SerialDriverVMT`
SerialDriver virtual methods table.
- `struct SerialDriver`
Full duplex serial driver class.

Functions

- void `sdInit` (void)
Serial Driver initialization.
- void `sdObjectInit` (`SerialDriver` *`sdp`, `qnotify_t` `inotify`, `qnotify_t` `onotify`)
Initializes a generic full duplex driver object.
- void `sdStart` (`SerialDriver` *`sdp`, const `SerialConfig` *`config`)
Configures and starts the driver.
- void `sdStop` (`SerialDriver` *`sdp`)
Stops the driver.
- void `sdIncomingData` (`SerialDriver` *`sdp`, `uint8_t` `b`)
Handles incoming data.
- `msg_t` `sdRequestData` (`SerialDriver` *`sdp`)
Handles outgoing data.

Defines

- `#define _serial_driver_methods _base_asynchronous_channel_methods`
SerialDriver specific methods.

Serial status flags

- `#define SD_PARITY_ERROR 32`
Parity error happened.
- `#define SD_FRAMING_ERROR 64`
Framing error happened.
- `#define SD_OVERRUN_ERROR 128`
Overflow happened.
- `#define SD_NOISE_ERROR 256`
Noise on the line.
- `#define SD_BREAK_DETECTED 512`
Break detected.

Serial configuration options

- `#define SERIAL_DEFAULT_BITRATE 38400`
Default bit rate.
- `#define SERIAL_BUFFERS_SIZE 16`
Serial buffers size.

Macro Functions

- `#define sdPutWouldBlock(sdp) chOQIsFull(&(sdp)->oqueue)`
Direct output check on a `SerialDriver`.
- `#define sdGetWouldBlock(sdp) chIQIsEmpty(&(sdp)->iqueue)`
Direct input check on a `SerialDriver`.
- `#define sdPut(sdp, b) chOQPut(&(sdp)->oqueue, b)`
Direct write to a `SerialDriver`.
- `#define sdPutTimeout(sdp, b, t) chOQPutTimeout(&(sdp)->oqueue, b, t)`
Direct write to a `SerialDriver` with timeout specification.
- `#define sdGet(sdp) chIQGet(&(sdp)->iqueue)`
Direct read from a `SerialDriver`.
- `#define sdGetTimeout(sdp, t) chIQGetTimeout(&(sdp)->iqueue, t)`
Direct read from a `SerialDriver` with timeout specification.
- `#define sdWrite(sdp, b, n) chOQWriteTimeout(&(sdp)->oqueue, b, n, TIME_INFINITE)`
Direct blocking write to a `SerialDriver`.
- `#define sdWriteTimeout(sdp, b, n, t) chOQWriteTimeout(&(sdp)->oqueue, b, n, t)`

- `#define sdAsynchronousWrite(sdp, b, n) chOQWriteTimeout(&(sdp)->oqueue, b, n, TIME_IMMEDIATE)`
Direct blocking write to a [SerialDriver](#) with timeout specification.
- `#define sdRead(sdp, b, n) chIQReadTimeout(&(sdp)->iqueue, b, n, TIME_INFINITE)`
Direct non-blocking write to a [SerialDriver](#).
- `#define sdReadTimeout(sdp, b, n, t) chIQReadTimeout(&(sdp)->iqueue, b, n, t)`
Direct blocking read from a [SerialDriver](#) with timeout specification.
- `#define sdAsynchronousRead(sdp, b, n) chIQReadTimeout(&(sdp)->iqueue, b, n, TIME_IMMEDIATE)`
Direct non-blocking read from a [SerialDriver](#).

Typedefs

- `typedef struct SerialDriver SerialDriver`
Structure representing a serial driver.

Enumerations

- `enum sdstate_t { SD_UNINIT = 0, SD_STOP = 1, SD_READY = 2 }`
Driver state machine possible states.

13.111 serial_lld.c File Reference

13.111.1 Detailed Description

Serial Driver subsystem low level driver source template.

```
#include "ch.h"
#include "hal.h"
```

Functions

- `void sd_lld_init (void)`
Low level serial driver initialization.
- `void sd_lld_start (SerialDriver *sdp, const SerialConfig *config)`
Low level serial driver configuration and (re)start.
- `void sd_lld_stop (SerialDriver *sdp)`
Low level serial driver stop.

Variables

- `SerialDriver SD1`
SD1 driver identifier.

13.112 serial_lld.h File Reference

13.112.1 Detailed Description

Serial Driver subsystem low level driver header template.

Data Structures

- struct `SerialConfig`
Generic Serial Driver configuration structure.

Functions

- void `sd_lld_init` (void)
Low level serial driver initialization.
- void `sd_lld_start` (`SerialDriver` *`sdp`, const `SerialConfig` *`config`)
Low level serial driver configuration and (re)start.
- void `sd_lld_stop` (`SerialDriver` *`sdp`)
Low level serial driver stop.

Defines

- #define `_serial_driver_data`
SerialDriver specific data.

Configuration options

- #define `PLATFORM_SERIAL_USE_SD1` FALSE
SD1 driver enable switch.

13.113 serial_usb.c File Reference

13.113.1 Detailed Description

Serial over USB Driver code.

```
#include "ch.h"
#include "hal.h"
```

Functions

- void `sdulinit` (void)
Serial Driver initialization.
- void `sduObjectInit` (`SerialUSBDriver` *`sdup`)
Initializes a generic full duplex driver object.
- void `sduStart` (`SerialUSBDriver` *`sdup`, const `SerialUSBConfig` *`config`)
Configures and starts the driver.
- void `sduStop` (`SerialUSBDriver` *`sdup`)
Stops the driver.
- void `sduConfigureHookI` (`SerialUSBDriver` *`sdup`)
USB device configured handler.
- `bool_t sduRequestsHook` (`USBDriver` *`usbp`)
Default requests hook.
- void `sduDataTransmitted` (`USBDriver` *`usbp`, `usbep_t` `ep`)
Default data transmitted callback.
- void `sduDataReceived` (`USBDriver` *`usbp`, `usbep_t` `ep`)
Default data received callback.
- void `sduInterruptTransmitted` (`USBDriver` *`usbp`, `usbep_t` `ep`)
Default data received callback.

13.114 serial_usb.h File Reference

13.114.1 Detailed Description

Serial over USB Driver macros and structures.

Data Structures

- struct `cdc_linecoding_t`
Type of Line Coding structure.
- struct `SerialUSBCConfig`
Serial over USB Driver configuration structure.
- struct `SerialUSBDriverVMT`
SerialDriver virtual methods table.
- struct `SerialUSBDriver`
Full duplex serial driver class.

Functions

- void `sduInit` (void)
Serial Driver initialization.
- void `sduObjectInit` (`SerialUSBDriver` *`sdup`)
Initializes a generic full duplex driver object.
- void `sduStart` (`SerialUSBDriver` *`sdup`, const `SerialUSBCConfig` *`config`)
Configures and starts the driver.
- void `sduStop` (`SerialUSBDriver` *`sdup`)
Stops the driver.
- void `sduConfigureHookI` (`SerialUSBDriver` *`sdup`)
USB device configured handler.
- `bool_t` `sduRequestsHook` (`USBDriver` *`usbp`)
Default requests hook.
- void `sduDataTransmitted` (`USBDriver` *`usbp`, `usbep_t` `ep`)
Default data transmitted callback.
- void `sduDataReceived` (`USBDriver` *`usbp`, `usbep_t` `ep`)
Default data received callback.
- void `sduInterruptTransmitted` (`USBDriver` *`usbp`, `usbep_t` `ep`)
Default data received callback.

Defines

- `#define _serial_usb_driver_data`
SerialDriver specific data.
- `#define _serial_usb_driver_methods` `_base_asynchronous_channel_methods`
SerialUSBDriver specific methods.

CDC specific messages.

- #define **CDC_SEND_ENCAPSULATED_COMMAND** 0x00
- #define **CDC_GET_ENCAPSULATED_RESPONSE** 0x01
- #define **CDC_SET_COMM_FEATURE** 0x02
- #define **CDC_GET_COMM_FEATURE** 0x03
- #define **CDC_CLEAR_COMM_FEATURE** 0x04
- #define **CDC_SET_AUX_LINE_STATE** 0x10
- #define **CDC_SET_HOOK_STATE** 0x11
- #define **CDC_PULSE_SETUP** 0x12
- #define **CDC_SEND_PULSE** 0x13
- #define **CDC_SET_PULSE_TIME** 0x14
- #define **CDC_RING_AUX_JACK** 0x15
- #define **CDC_SET_LINE_CODING** 0x20
- #define **CDC_GET_LINE_CODING** 0x21
- #define **CDC_SET_CONTROL_LINE_STATE** 0x22
- #define **CDC_SEND_BREAK** 0x23
- #define **CDC_SET_RINGER_PARMS** 0x30
- #define **CDC_GET_RINGER_PARMS** 0x31
- #define **CDC_SET_OPERATION_PARMS** 0x32
- #define **CDC_GET_OPERATION_PARMS** 0x33

Line Control bit definitions.

- #define **LC_STOP_1** 0
- #define **LC_STOP_1P5** 1
- #define **LC_STOP_2** 2
- #define **LC_PARITY_NONE** 0
- #define **LC_PARITY_ODD** 1
- #define **LC_PARITY_EVEN** 2
- #define **LC_PARITY_MARK** 3
- #define **LC_PARITY_SPACE** 4

SERIAL_USB configuration options

- #define **SERIAL_USB_BUFFERS_SIZE** 256
Serial over USB buffers size.

Typedefs

- typedef struct **SerialUSBDriver** **SerialUSBDriver**
Structure representing a serial over USB driver.

Enumerations

- enum **sdustate_t** { **SDU_UNINIT** = 0, **SDU_STOP** = 1, **SDU_READY** = 2 }
- Driver state machine possible states.*

13.115 shell.c File Reference

13.115.1 Detailed Description

```
Simple CLI shell code. #include <string.h>
#include "ch.h"
#include "hal.h"
#include "shell.h"
#include "chprintf.h"
```

Functions

- void **shellInit** (void)
Shell manager initialization.
- void **shellExit** (msg_t msg)
Terminates the shell.
- Thread * **shellCreate** (const ShellConfig *scp, size_t size, tprio_t prio)
Spawns a new shell.
- Thread * **shellCreateStatic** (const ShellConfig *scp, void *wsp, size_t size, tprio_t prio)
Create statically allocated shell thread.
- bool_t **shellGetLine** (BaseSequentialStream *chp, char *line, unsigned size)
Reads a whole line from the input channel.

Variables

- EventSource **shell_terminated**
Shell termination event source.

13.116 shell.h File Reference

13.116.1 Detailed Description

Simple CLI shell header.

Data Structures

- struct **ShellCommand**
Custom command entry type.
- struct **ShellConfig**
Shell descriptor type.

Functions

- void **shellInit** (void)
Shell manager initialization.
- void **shellExit** (msg_t msg)
Terminates the shell.
- Thread * **shellCreate** (const ShellConfig *scp, size_t size, tprio_t prio)
Spawns a new shell.
- Thread * **shellCreateStatic** (const ShellConfig *scp, void *wsp, size_t size, tprio_t prio)
Create statically allocated shell thread.
- bool_t **shellGetLine** (BaseSequentialStream *chp, char *line, unsigned size)
Reads a whole line from the input channel.

Defines

- #define **SHELL_MAX_LINE_LENGTH** 64
Shell maximum input line length.
- #define **SHELL_MAX_ARGUMENTS** 4
Shell maximum arguments per command.

Typedefs

- `typedef void(* shellcmd_t)(BaseSequentialStream *chp, int argc, char *argv[])`
Command handler function type.

13.117 spi.c File Reference

13.117.1 Detailed Description

SPI Driver code. #include "ch.h"
#include "hal.h"

Functions

- `void spilinit (void)`
SPI Driver initialization.
- `void spiObjectInit (SPIDriver *spip)`
Initializes the standard part of a `SPIDriver` structure.
- `void spiStart (SPIDriver *spip, const SPIConfig *config)`
Configures and activates the SPI peripheral.
- `void spiStop (SPIDriver *spip)`
Deactivates the SPI peripheral.
- `void spiSelect (SPIDriver *spip)`
Asserts the slave select signal and prepares for transfers.
- `void spiUnselect (SPIDriver *spip)`
Deasserts the slave select signal.
- `void spiStartIgnore (SPIDriver *spip, size_t n)`
Ignores data on the SPI bus.
- `void spiStartExchange (SPIDriver *spip, size_t n, const void *txbuf, void *rxbuf)`
Exchanges data on the SPI bus.
- `void spiStartSend (SPIDriver *spip, size_t n, const void *txbuf)`
Sends data over the SPI bus.
- `void spiStartReceive (SPIDriver *spip, size_t n, void *rxbuf)`
Receives data from the SPI bus.
- `void spignore (SPIDriver *spip, size_t n)`
Ignores data on the SPI bus.
- `void spiExchange (SPIDriver *spip, size_t n, const void *txbuf, void *rxbuf)`
Exchanges data on the SPI bus.
- `void spiSend (SPIDriver *spip, size_t n, const void *txbuf)`
Sends data over the SPI bus.
- `void spiReceive (SPIDriver *spip, size_t n, void *rxbuf)`
Receives data from the SPI bus.
- `void spiAcquireBus (SPIDriver *spip)`
Gains exclusive access to the SPI bus.
- `void spiReleaseBus (SPIDriver *spip)`
Releases exclusive access to the SPI bus.

13.118 spi.h File Reference

13.118.1 Detailed Description

SPI Driver macros and structures. #include "spi_lld.h"

Functions

- void **spiInit** (void)
SPI Driver initialization.
- void **spiObjectInit** (SPIDriver *spip)
Initializes the standard part of a SPIDriver structure.
- void **spiStart** (SPIDriver *spip, const SPIConfig *config)
Configures and activates the SPI peripheral.
- void **spiStop** (SPIDriver *spip)
Deactivates the SPI peripheral.
- void **spiSelect** (SPIDriver *spip)
Asserts the slave select signal and prepares for transfers.
- void **spiUnselect** (SPIDriver *spip)
Deasserts the slave select signal.
- void **spiStartIgnore** (SPIDriver *spip, size_t n)
Ignores data on the SPI bus.
- void **spiStartExchange** (SPIDriver *spip, size_t n, const void *txbuf, void *rxbuf)
Exchanges data on the SPI bus.
- void **spiStartSend** (SPIDriver *spip, size_t n, const void *txbuf)
Sends data over the SPI bus.
- void **spiStartReceive** (SPIDriver *spip, size_t n, void *rxbuf)
Receives data from the SPI bus.

Defines

SPI configuration options

- #define **SPI_USE_WAIT** TRUE
Enables synchronous APIs.
- #define **SPI_USE_MUTUAL_EXCLUSION** TRUE
Enables the spiAcquireBus () and spiReleaseBus () APIs.

Macro Functions

- #define **spiSelectl**(spip)
Asserts the slave select signal and prepares for transfers.
- #define **spiUnselectl**(spip)
Deasserts the slave select signal.
- #define **spiStartIgnorel**(spip, n)
Ignores data on the SPI bus.
- #define **spiStartExchangel**(spip, n, txbuf, rxbuf)
Exchanges data on the SPI bus.
- #define **spiStartSendl**(spip, n, txbuf)
Sends data over the SPI bus.
- #define **spiStartReceivel**(spip, n, rxbuf)
Receives data from the SPI bus.
- #define **spiPolledExchange**(spip, frame) spi_lld_polled_exchange(spip, frame)
Exchanges one frame using a polled wait.

Low Level driver helper macros

- `#define _spi_wait_s(spip)`
Waits for operation completion.
- `#define _spi_wakeup_isr(spip)`
Wakes up the waiting thread.
- `#define _spi_isr_code(spip)`
Common ISR code.

Enumerations

- enum `spistate_t` {

 `SPI_UNINIT = 0, SPI_STOP = 1, SPI_READY = 2, SPI_ACTIVE = 3,`

 `SPI_COMPLETE = 4 }`

Driver state machine possible states.

13.119 spi_lld.c File Reference

13.119.1 Detailed Description

SPI Driver subsystem low level driver source template.

```
#include "ch.h"
#include "hal.h"
```

Functions

- `void spi_lld_init (void)`
Low level SPI driver initialization.
- `void spi_lld_start (SPIDriver *spip)`
Configures and activates the SPI peripheral.
- `void spi_lld_stop (SPIDriver *spip)`
Deactivates the SPI peripheral.
- `void spi_lld_select (SPIDriver *spip)`
Asserts the slave select signal and prepares for transfers.
- `void spi_lld_unselect (SPIDriver *spip)`
Deasserts the slave select signal.
- `void spi_lld_ignore (SPIDriver *spip, size_t n)`
Ignores data on the SPI bus.
- `void spi_lld_exchange (SPIDriver *spip, size_t n, const void *txbuf, void *rxbuf)`
Exchanges data on the SPI bus.
- `void spi_lld_send (SPIDriver *spip, size_t n, const void *txbuf)`
Sends data over the SPI bus.
- `void spi_lld_receive (SPIDriver *spip, size_t n, void *rxbuf)`
Receives data from the SPI bus.
- `uint16_t spi_lld_polled_exchange (SPIDriver *spip, uint16_t frame)`
Exchanges one frame using a polled wait.

Variables

- `SPIDriver SPID1`
SPI1 driver identifier.

13.120 spi_lld.h File Reference

13.120.1 Detailed Description

SPI Driver subsystem low level driver header template.

Data Structures

- struct [SPIConfig](#)
Driver configuration structure.
- struct [SPIDriver](#)
Structure representing an SPI driver.

Functions

- void [spi_lld_init](#) (void)
Low level SPI driver initialization.
- void [spi_lld_start](#) (SPIDriver *spip)
Configures and activates the SPI peripheral.
- void [spi_lld_stop](#) (SPIDriver *spip)
Deactivates the SPI peripheral.
- void [spi_lld_select](#) (SPIDriver *spip)
Asserts the slave select signal and prepares for transfers.
- void [spi_lld_unselect](#) (SPIDriver *spip)
Deasserts the slave select signal.
- void [spi_lld_ignore](#) (SPIDriver *spip, size_t n)
Ignores data on the SPI bus.
- void [spi_lld_exchange](#) (SPIDriver *spip, size_t n, const void *txbuf, void *rxbuf)
Exchanges data on the SPI bus.
- void [spi_lld_send](#) (SPIDriver *spip, size_t n, const void *txbuf)
Sends data over the SPI bus.
- void [spi_lld_receive](#) (SPIDriver *spip, size_t n, void *rxbuf)
Receives data from the SPI bus.
- uint16_t [spi_lld_polled_exchange](#) (SPIDriver *spip, uint16_t frame)
Exchanges one frame using a polled wait.

Defines

Configuration options

- #define [PLATFORM_SPI_USE_SPI1](#) FALSE
SPI driver enable switch.

Typedefs

- typedef struct [SPIDriver](#) SPIDriver
Type of a structure representing an SPI driver.
- typedef void(* [spicallback_t](#))(SPIDriver *spip)
SPI notification callback type.

13.121 test.c File Reference

13.121.1 Detailed Description

Tests support code.

```
#include "ch.h"
#include "hal.h"
#include "test.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_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.
- void **test_cpu_pulse** (unsigned duration)
CPU pulse.
- **systime_t test_wait_tick** (void)
Delays execution until next system time tick.
- void **test_start_timer** (unsigned ms)
Starts the test timer.
- **msg_t TestThread** (void *p)
Test execution thread function.

Variables

- **bool_t test_timer_done**
Set to TRUE when the test timer reaches its deadline.

13.122 test.h File Reference

13.122.1 Detailed Description

Tests support header.

Data Structures

- struct **testcase**

Structure representing a test case.

Functions

- **msg_t 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.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 test_cpu_pulse (unsigned duration)**
CPU pulse.

Defines

- **#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.

13.123 testbmk.c File Reference

13.123.1 Detailed Description

Kernel Benchmarks source file. Kernel Benchmarks #include "ch.h"
#include "test.h"

Variables

- ROMCONST struct testcase *ROMCONST patternbmk []
Test sequence for benchmarks.

13.123.2 Variable Documentation

13.123.2.1 ROMCONST struct testcase* ROMCONST patternbmk[]

Initial value:

```
{  
    &testbmk1,  
    &testbmk2,  
    &testbmk3,  
    &testbmk4,  
    &testbmk5,  
    &testbmk6,  
    &testbmk7,  
    &testbmk8,  
  
    &testbmk9,  
  
    &testbmk10,  
  
    &testbmk11,  
  
    &testbmk12,  
  
    &testbmk13,  
  
    NULL  
}
```

Test sequence for benchmarks.

13.124 testbmk.h File Reference

13.124.1 Detailed Description

Kernel Benchmarks header file.

Variables

- ROMCONST struct testcase *ROMCONST patternbmk []
Test sequence for benchmarks.

13.124.2 Variable Documentation

13.124.2.1 ROMCONST struct testcase* ROMCONST patternbmk[]

Test sequence for benchmarks.

13.125 testdyn.c File Reference

13.125.1 Detailed Description

Dynamic thread APIs test source file.

```
#include "ch.h"
#include "test.h"
```

Variables

- ROMCONST struct testcase *ROMCONST patterndyn []
Test sequence for dynamic APIs.

13.125.2 Variable Documentation

13.125.2.1 ROMCONST struct testcase* ROMCONST patterndyn[]

Initial value:

```
{
    &testdyn1,
    &testdyn2,
    &testdyn3,
    NULL
}
```

Test sequence for dynamic APIs.

13.126 testdyn.h File Reference

13.126.1 Detailed Description

Dynamic thread APIs test header file.

Variables

- ROMCONST struct testcase *ROMCONST patterndyn []
Test sequence for dynamic APIs.

13.126.2 Variable Documentation

13.126.2.1 ROMCONST struct testcase* ROMCONST patterndyn[]

Test sequence for dynamic APIs.

13.127 testevt.c File Reference

13.127.1 Detailed Description

Events test source file.

```
#include "ch.h"  
#include "test.h"
```

Variables

- ROMCONST struct testcase *ROMCONST patternevt []
Test sequence for events.

13.127.2 Variable Documentation

13.127.2.1 ROMCONST struct testcase* ROMCONST patternevt[]

Initial value:

```
{  
    &testevt1,  
    &testevt2,  
  
    &testevt3,  
  
    NULL  
}
```

Test sequence for events.

13.128 testevt.h File Reference

13.128.1 Detailed Description

Events test header file.

Variables

- ROMCONST struct testcase *ROMCONST patternevt []
Test sequence for events.

13.128.2 Variable Documentation

13.128.2.1 ROMCONST struct testcase* ROMCONST patternevt[]

Test sequence for events.

13.129 testheap.c File Reference

13.129.1 Detailed Description

```
Heap test source file. #include "ch.h"  
#include "test.h"
```

Variables

- ROMCONST struct testcase *ROMCONST patternheap []
Test sequence for heap.

13.129.2 Variable Documentation

13.129.2.1 ROMCONST struct testcase* ROMCONST patternheap[]

Initial value:

```
{  
    &testheap1,  
    NULL  
}
```

Test sequence for heap.

13.130 testheap.h File Reference

13.130.1 Detailed Description

Heap header file.

Variables

- ROMCONST struct testcase *ROMCONST patternheap []
Test sequence for heap.

13.130.2 Variable Documentation

13.130.2.1 ROMCONST struct testcase* ROMCONST patternheap[]

Test sequence for heap.

13.131 testmbox.c File Reference

13.131.1 Detailed Description

```
Mailboxes test source file. #include "ch.h"  
#include "test.h"
```

Variables

- ROMCONST struct `testcase` *ROMCONST `patternmbox` []
Test sequence for mailboxes.

13.131.2 Variable Documentation

13.131.2.1 ROMCONST struct `testcase`* ROMCONST `patternmbox`[]

Initial value:

```
{  
    &testmbox1,  
    NULL  
}
```

Test sequence for mailboxes.

13.132 testmbox.h File Reference

13.132.1 Detailed Description

Mailboxes header file.

Variables

- ROMCONST struct `testcase` *ROMCONST `patternmbox` []
Test sequence for mailboxes.

13.132.2 Variable Documentation

13.132.2.1 ROMCONST struct `testcase`* ROMCONST `patternmbox`[]

Test sequence for mailboxes.

13.133 testmsg.c File Reference

13.133.1 Detailed Description

```
Messages test source file.  
#include "ch.h"  
#include "test.h"
```

Variables

- ROMCONST struct `testcase` *ROMCONST `patternmsg` []
Test sequence for messages.

13.133.2 Variable Documentation

13.133.2.1 ROMCONST struct testcase* ROMCONST patternmsg[]

Initial value:

```
{  
    &testmsg1,  
    NULL  
}
```

Test sequence for messages.

13.134 testmsg.h File Reference

13.134.1 Detailed Description

Messages header file.

Variables

- ROMCONST struct [testcase](#) *ROMCONST [patternmsg](#) []
Test sequence for messages.

13.134.2 Variable Documentation

13.134.2.1 ROMCONST struct testcase* ROMCONST patternmsg[]

Test sequence for messages.

13.135 testmtx.c File Reference

13.135.1 Detailed Description

```
Mutexes and CondVars test source file.  
#include "ch.h"  
#include "test.h"
```

Variables

- ROMCONST struct [testcase](#) *ROMCONST [patternmtx](#) []
Test sequence for mutexes.

13.135.2 Variable Documentation

13.135.2.1 ROMCONST struct testcase* ROMCONST patternmtx[]

Initial value:

```
{  
    &testmtx1,  
  
    &testmtx2,  
    &testmtx3,  
  
    &testmtx4,  
    &testmtx5,  
  
    &testmtx6,  
    &testmtx7,  
    &testmtx8,  
  
    NULL  
}
```

Test sequence for mutexes.

13.136 testmtx.h File Reference

13.136.1 Detailed Description

Mutexes and CondVars test header file.

Variables

- ROMCONST struct [testcase](#) *ROMCONST [patternmtx](#) []
Test sequence for mutexes.

13.136.2 Variable Documentation

13.136.2.1 ROMCONST struct [testcase](#)* ROMCONST [patternmtx](#)[]

Test sequence for mutexes.

13.137 testpools.c File Reference

13.137.1 Detailed Description

```
#include "ch.h"  
#include "test.h"
```

13.138 testpools.h File Reference

13.138.1 Detailed Description

Memory Pools test header file.

13.139 testqueues.c File Reference

13.139.1 Detailed Description

```
I/O Queues test source file. #include "ch.h"  
#include "test.h"
```

Variables

- ROMCONST struct testcase *ROMCONST patternqueues []
Test sequence for queues.

13.139.2 Variable Documentation

13.139.2.1 ROMCONST struct testcase* ROMCONST patternqueues[]

Initial value:

```
{  
    &testqueues1,  
    &testqueues2,  
    NULL  
}
```

Test sequence for queues.

13.140 testqueues.h File Reference

13.140.1 Detailed Description

I/O Queues test header file.

Variables

- ROMCONST struct testcase *ROMCONST patternqueues []
Test sequence for queues.

13.140.2 Variable Documentation

13.140.2.1 ROMCONST struct testcase* ROMCONST patternqueues[]

Test sequence for queues.

13.141 testsem.c File Reference

13.141.1 Detailed Description

```
Semaphores test source file. #include "ch.h"  
#include "test.h"
```

Variables

- ROMCONST struct `testcase` *ROMCONST `patternsem` []
Test sequence for semaphores.

13.141.2 Variable Documentation

13.141.2.1 ROMCONST struct `testcase`* ROMCONST `patternsem`[]

Initial value:

```
{  
    &testsem1,  
    &testsem2,  
  
    &testsem3,  
  
    &testsem4,  
  
    NULL  
}
```

Test sequence for semaphores.

13.142 testsem.h File Reference

13.142.1 Detailed Description

Semaphores test header file.

Variables

- ROMCONST struct `testcase` *ROMCONST `patternsem` []
Test sequence for semaphores.

13.142.2 Variable Documentation

13.142.2.1 ROMCONST struct `testcase`* ROMCONST `patternsem`[]

Test sequence for semaphores.

13.143 testthd.c File Reference

13.143.1 Detailed Description

Threads and Scheduler test source file.
#include "ch.h"
#include "test.h"

Variables

- ROMCONST struct `testcase` *ROMCONST `patternthd` []
Test sequence for threads.

13.143.2 Variable Documentation

13.143.2.1 ROMCONST struct testcase* ROMCONST patternhd[]

Initial value:

```
{  
    &testthd1,  
    &testthd2,  
    &testthd3,  
    &testthd4,  
    NULL  
}
```

Test sequence for threads.

13.144 testthd.h File Reference

13.144.1 Detailed Description

Threads and Scheduler test header file.

Variables

- ROMCONST struct [testcase](#) *ROMCONST [patternhd](#) []
Test sequence for threads.

13.144.2 Variable Documentation

13.144.2.1 ROMCONST struct testcase* ROMCONST patternhd[]

Test sequence for threads.

13.145 tm.c File Reference

13.145.1 Detailed Description

Time Measurement driver code.

```
#include "ch.h"  
#include "hal.h"
```

Functions

- void [tmInit](#) (void)
Initializes the Time Measurement unit.
- void [tmObjectInit](#) ([TimeMeasurement](#) *tmp)
Initializes a [TimeMeasurement](#) object.

13.146 tm.h File Reference

13.146.1 Detailed Description

Time Measurement driver header.

Data Structures

- struct **TimeMeasurement**

Time Measurement structure.

Functions

- void **tmInit** (void)
Initializes the Time Measurement unit.
- void **tmObjectInit** (**TimeMeasurement** *tmp)
*Initializes a **TimeMeasurement** object.*

Defines

- #define **tmStartMeasurement**(tmp) (tmp)->start(tmp)
Starts a measurement.
- #define **tmStopMeasurement**(tmp) (tmp)->stop(tmp)
Stops a measurement.

Typedefs

- typedef struct **TimeMeasurement** **TimeMeasurement**
Type of a Time Measurement object.

13.147 uart.c File Reference

13.147.1 Detailed Description

UART Driver code. #include "ch.h"
#include "hal.h"

Functions

- void **uartInit** (void)
UART Driver initialization.
- void **uartObjectInit** (**UARTDriver** *uartp)
*Initializes the standard part of a **UARTDriver** structure.*
- void **uartStart** (**UARTDriver** *uartp, const **UARTConfig** *config)
Configures and activates the UART peripheral.
- void **uartStop** (**UARTDriver** *uartp)
Deactivates the UART peripheral.
- void **uartStartSend** (**UARTDriver** *uartp, size_t n, const void *txbuf)
Starts a transmission on the UART peripheral.
- void **uartStartSendl** (**UARTDriver** *uartp, size_t n, const void *txbuf)
Starts a transmission on the UART peripheral.
- size_t **uartStopSend** (**UARTDriver** *uartp)
Stops any ongoing transmission.
- size_t **uartStopSendl** (**UARTDriver** *uartp)
Stops any ongoing transmission.

- void `uartStartReceive (UARTDriver *uartp, size_t n, void *rdbuf)`
Starts a receive operation on the UART peripheral.
- void `uartStartReceive1 (UARTDriver *uartp, size_t n, void *rdbuf)`
Starts a receive operation on the UART peripheral.
- size_t `uartStopReceive (UARTDriver *uartp)`
Stops any ongoing receive operation.
- size_t `uartStopReceive1 (UARTDriver *uartp)`
Stops any ongoing receive operation.

13.148 uart.h File Reference

13.148.1 Detailed Description

UART Driver macros and structures. #include "uart_llld.h"

Functions

- void `uartInit (void)`
UART Driver initialization.
- void `uartObjectInit (UARTDriver *uartp)`
Initializes the standard part of a `UARTDriver` structure.
- void `uartStart (UARTDriver *uartp, const UARTConfig *config)`
Configures and activates the UART peripheral.
- void `uartStop (UARTDriver *uartp)`
Deactivates the UART peripheral.
- void `uartStartSend (UARTDriver *uartp, size_t n, const void *txbuf)`
Starts a transmission on the UART peripheral.
- void `uartStartSend1 (UARTDriver *uartp, size_t n, const void *txbuf)`
Starts a transmission on the UART peripheral.
- size_t `uartStopSend (UARTDriver *uartp)`
Stops any ongoing transmission.
- size_t `uartStopSend1 (UARTDriver *uartp)`
Stops any ongoing transmission.
- void `uartStartReceive (UARTDriver *uartp, size_t n, void *rdbuf)`
Starts a receive operation on the UART peripheral.
- void `uartStartReceive1 (UARTDriver *uartp, size_t n, void *rdbuf)`
Starts a receive operation on the UART peripheral.
- size_t `uartStopReceive (UARTDriver *uartp)`
Stops any ongoing receive operation.
- size_t `uartStopReceive1 (UARTDriver *uartp)`
Stops any ongoing receive operation.

Defines

UART status flags

- #define `UART_NO_ERROR` 0
No pending conditions.
- #define `UART_PARITY_ERROR` 4
Parity error happened.
- #define `UART_FRAMING_ERROR` 8

- `#define UART_OVERRUN_ERROR 16`
Framing error happened.
- `#define UART_NOISE_ERROR 32`
Overflow happened.
- `#define UART_BREAK_DETECTED 64`
Noise on the line.
- `#define UART_BREAK_DETECTED 64`
Break detected.

Enumerations

- enum `uartstate_t { UART_UNINIT = 0, UART_STOP = 1, UART_READY = 2 }`
Driver state machine possible states.
- enum `uarttxstate_t { UART_TX_IDLE = 0, UART_TX_ACTIVE = 1, UART_TX_COMPLETE = 2 }`
Transmitter state machine states.
- enum `uartrxstate_t { UART_RX_IDLE = 0, UART_RX_ACTIVE = 1, UART_RX_COMPLETE = 2 }`
Receiver state machine states.

13.149 uart_lld.c File Reference

13.149.1 Detailed Description

UART Driver subsystem low level driver source template.

```
#include "ch.h"
#include "hal.h"
```

Functions

- void `uart_lld_init (void)`
Low level UART driver initialization.
- void `uart_lld_start (UARTDriver *uartp)`
Configures and activates the UART peripheral.
- void `uart_lld_stop (UARTDriver *uartp)`
Deactivates the UART peripheral.
- void `uart_lld_start_send (UARTDriver *uartp, size_t n, const void *txbuf)`
Starts a transmission on the UART peripheral.
- size_t `uart_lld_stop_send (UARTDriver *uartp)`
Stops any ongoing transmission.
- void `uart_lld_start_receive (UARTDriver *uartp, size_t n, void *rdbuf)`
Starts a receive operation on the UART peripheral.
- size_t `uart_lld_stop_receive (UARTDriver *uartp)`
Stops any ongoing receive operation.

Variables

- `UARTDriver UARTD1`
UART1 driver identifier.

13.150 uart_lld.h File Reference

13.150.1 Detailed Description

UART Driver subsystem low level driver header template.

Data Structures

- struct **UARTConfig**
Driver configuration structure.
- struct **UARTDriver**
Structure representing an UART driver.

Functions

- void **uart_ll_init** (void)
Low level UART driver initialization.
- void **uart_ll_start** (**UARTDriver** *uartp)
Configures and activates the UART peripheral.
- void **uart_ll_stop** (**UARTDriver** *uartp)
Deactivates the UART peripheral.
- void **uart_ll_start_send** (**UARTDriver** *uartp, size_t n, const void *txbuf)
Starts a transmission on the UART peripheral.
- size_t **uart_ll_stop_send** (**UARTDriver** *uartp)
Stops any ongoing transmission.
- void **uart_ll_start_receive** (**UARTDriver** *uartp, size_t n, void *rxbuf)
Starts a receive operation on the UART peripheral.
- size_t **uart_ll_stop_receive** (**UARTDriver** *uartp)
Stops any ongoing receive operation.

Defines

Configuration options

- #define **PLATFORM_UART_USE_UART1** FALSE
UART driver enable switch.

Typedefs

- typedef uint32_t **uartflags_t**
UART driver condition flags type.
- typedef struct **UARTDriver** **UARTDriver**
Type of structure representing an UART driver.
- typedef void(* **uartcb_t**)(**UARTDriver** *uartp)
Generic UART notification callback type.
- typedef void(* **uartccb_t**)(**UARTDriver** *uartp, uint16_t c)
Character received UART notification callback type.
- typedef void(* **uar tcb_t**)(**UARTDriver** *uartp, **uartflags_t** e)
Receive error UART notification callback type.

13.151 usb.c File Reference

13.151.1 Detailed Description

USB Driver code.

```
#include <string.h>
#include "ch.h"
#include "hal.h"
#include "usb.h"
```

Functions

- void **usbInit** (void)

USB Driver initialization.
- void **usbObjectInit** (**USBDriver** *usbp)

*Initializes the standard part of a **USBDriver** structure.*
- void **usbStart** (**USBDriver** *usbp, const **USBConfig** *config)

Configures and activates the USB peripheral.
- void **usbStop** (**USBDriver** *usbp)

Deactivates the USB peripheral.
- void **usbInitEndpoint** (**USBDriver** *usbp, **usbep_t** ep, const **USBEndpointConfig** *epcp)

Enables an endpoint.
- void **usbDisableEndpoints** (**USBDriver** *usbp)

Disables all the active endpoints.
- void **usbPrepareReceive** (**USBDriver** *usbp, **usbep_t** ep, **uint8_t** *buf, **size_t** n)

Prepares for a receive transaction on an OUT endpoint.
- void **usbPrepareTransmit** (**USBDriver** *usbp, **usbep_t** ep, const **uint8_t** *buf, **size_t** n)

Prepares for a transmit transaction on an IN endpoint.
- void **usbPrepareQueuedReceive** (**USBDriver** *usbp, **usbep_t** ep, **InputQueue** *iqp, **size_t** n)

Prepares for a receive transaction on an OUT endpoint.
- void **usbPrepareQueuedTransmit** (**USBDriver** *usbp, **usbep_t** ep, **OutputQueue** *oqp, **size_t** n)

Prepares for a transmit transaction on an IN endpoint.
- **bool_t** **usbStartReceivel** (**USBDriver** *usbp, **usbep_t** ep)

Starts a receive transaction on an OUT endpoint.
- **bool_t** **usbStartTransmitl** (**USBDriver** *usbp, **usbep_t** ep)

Starts a transmit transaction on an IN endpoint.
- **bool_t** **usbStallReceivel** (**USBDriver** *usbp, **usbep_t** ep)

Stalls an OUT endpoint.
- **bool_t** **usbStallTransmitl** (**USBDriver** *usbp, **usbep_t** ep)

Stalls an IN endpoint.
- void **_usb_reset** (**USBDriver** *usbp)

USB reset routine.
- void **_usb_ep0setup** (**USBDriver** *usbp, **usbep_t** ep)

Default EP0 SETUP callback.
- void **_usb_ep0in** (**USBDriver** *usbp, **usbep_t** ep)

Default EP0 IN callback.
- void **_usb_ep0out** (**USBDriver** *usbp, **usbep_t** ep)

Default EP0 OUT callback.

13.152 usb.h File Reference

13.152.1 Detailed Description

USB Driver macros and structures. #include "usb_lld.h"

Data Structures

- struct [USBDescriptor](#)
Type of an USB descriptor.

Functions

- void [usbInit](#) (void)
USB Driver initialization.
- void [usbObjectInit](#) ([USBDriver](#) *usbp)
Initializes the standard part of a [USBDriver](#) structure.
- void [usbStart](#) ([USBDriver](#) *usbp, const [USBConfig](#) *config)
Configures and activates the USB peripheral.
- void [usbStop](#) ([USBDriver](#) *usbp)
Deactivates the USB peripheral.
- void [usbInitEndpointl](#) ([USBDriver](#) *usbp, [usbep_t](#) ep, const [USBEndpointConfig](#) *epcp)
Enables an endpoint.
- void [usbDisableEndpointsl](#) ([USBDriver](#) *usbp)
Disables all the active endpoints.
- void [usbPrepareReceive](#) ([USBDriver](#) *usbp, [usbep_t](#) ep, [uint8_t](#) *buf, [size_t](#) n)
Prepares for a receive transaction on an OUT endpoint.
- void [usbPrepareTransmit](#) ([USBDriver](#) *usbp, [usbep_t](#) ep, const [uint8_t](#) *buf, [size_t](#) n)
Prepares for a transmit transaction on an IN endpoint.
- void [usbPrepareQueuedReceive](#) ([USBDriver](#) *usbp, [usbep_t](#) ep, [InputQueue](#) *iqp, [size_t](#) n)
Prepares for a receive transaction on an OUT endpoint.
- void [usbPrepareQueuedTransmit](#) ([USBDriver](#) *usbp, [usbep_t](#) ep, [OutputQueue](#) *oqp, [size_t](#) n)
Prepares for a transmit transaction on an IN endpoint.
- [bool_t](#) [usbStartReceivel](#) ([USBDriver](#) *usbp, [usbep_t](#) ep)
Starts a receive transaction on an OUT endpoint.
- [bool_t](#) [usbStartTransmitl](#) ([USBDriver](#) *usbp, [usbep_t](#) ep)
Starts a transmit transaction on an IN endpoint.
- [bool_t](#) [usbStallReceivel](#) ([USBDriver](#) *usbp, [usbep_t](#) ep)
Stalls an OUT endpoint.
- [bool_t](#) [usbStallTransmitl](#) ([USBDriver](#) *usbp, [usbep_t](#) ep)
Stalls an IN endpoint.
- void [_usb_reset](#) ([USBDriver](#) *usbp)
USB reset routine.
- void [_usb_ep0setup](#) ([USBDriver](#) *usbp, [usbep_t](#) ep)
Default EP0 SETUP callback.
- void [_usb_ep0in](#) ([USBDriver](#) *usbp, [usbep_t](#) ep)
Default EP0 IN callback.
- void [_usb_ep0out](#) ([USBDriver](#) *usbp, [usbep_t](#) ep)
Default EP0 OUT callback.

Defines

Helper macros for USB descriptors

- `#define USB_DESC_INDEX(i) ((uint8_t)(i))`
Helper macro for index values into descriptor strings.
- `#define USB_DESC_BYTE(b) ((uint8_t)(b))`
Helper macro for byte values into descriptor strings.
- `#define USB_DESC_WORD(w)`
Helper macro for word values into descriptor strings.
- `#define USB_DESC_BCD(bcd)`
Helper macro for BCD values into descriptor strings.
- `#define USB_DESC_DEVICE(bcdUSB, bDeviceClass, bDeviceSubClass, bDeviceProtocol, bMaxPacketSize, idVendor, idProduct, bcdDevice, iManufacturer, iProduct, iSerialNumber, bNumConfigurations)`
Device Descriptor helper macro.
- `#define USB_DESC_CONFIGURATION(wTotalLength, bNumInterfaces, bConfigurationValue, iConfiguration, bmAttributes, bMaxPower)`
Configuration Descriptor helper macro.
- `#define USB_DESC_INTERFACE(blInterfaceNumber, bAlternateSetting, bNumEndpoints, blInterfaceClass, blInterfaceSubClass, blInterfaceProtocol, iInterface)`
Interface Descriptor helper macro.
- `#define USB_DESC_INTERFACE_ASSOCIATION(bFirstInterface, blInterfaceCount, bFunctionClass, bFunctionSubClass, bFunctionProtocol, iInterface)`
Interface Association Descriptor helper macro.
- `#define USB_DESC_ENDPOINT(bEndpointAddress, bmAttributes, wMaxPacketSize, blInterval)`
Endpoint Descriptor helper macro.

Endpoint types and settings

- `#define USB_EP_MODE_TYPE 0x0003`
- `#define USB_EP_MODE_TYPE_CTRL 0x0000`
- `#define USB_EP_MODE_TYPE_ISOC 0x0001`
- `#define USB_EP_MODE_TYPE_BULK 0x0002`
- `#define USB_EP_MODE_TYPE_INTR 0x0003`
- `#define USB_EP_MODE_LINEAR_BUFFER 0x0000`
- `#define USB_EP_MODE_QUEUE_BUFFER 0x0010`

Macro Functions

- `#define usbGetDriverState(usbp) ((usbp)->state)`
Returns the driver state.
- `#define usbFetchWord(p) ((uint16_t)*(p) | ((uint16_t)*((p) + 1) << 8))`
Fetches a 16 bits word value from an USB message.
- `#define usbConnectBus(usbp) usb_ll_connect_bus(usbp)`
Connects the USB device.
- `#define usbDisconnectBus(usbp) usb_ll_disconnect_bus(usbp)`
Disconnect the USB device.
- `#define usbGetFrameNumber(usbp) usb_ll_get_frame_number(usbp)`
Returns the current frame number.
- `#define usbGetTransmitStatus(usbp, ep) ((usbp)->transmitting & (1 << (ep)))`
Returns the status of an IN endpoint.
- `#define usbGetReceiveStatus(usbp, ep) ((usbp)->receiving & (1 << (ep)))`
Returns the status of an OUT endpoint.
- `#define usbGetReceiveTransactionSize(usbp, ep) usb_ll_get_transaction_size(usbp, ep)`
Returns the exact size of a receive transaction.
- `#define usbSetupTransfer(usbp, buf, n, endcb)`
Request transfer setup.
- `#define usbReadSetup(usbp, ep, buf) usb_ll_read_setup(usbp, ep, buf)`
Reads a setup packet from the dedicated packet buffer.

Low Level driver helper macros

- `#define _usb_isr_invoke_event_cb(usbp, evt)`
Common ISR code, usb event callback.
- `#define _usb_isr_invoke_sof_cb(usbp)`
Common ISR code, SOF callback.
- `#define _usb_isr_invoke_setup_cb(usbp, ep)`
Common ISR code, setup packet callback.
- `#define _usb_isr_invoke_in_cb(usbp, ep)`
Common ISR code, IN endpoint callback.
- `#define _usb_isr_invoke_out_cb(usbp, ep)`
Common ISR code, OUT endpoint event.

Typedefs

- `typedef struct USBDriver USBDriver`
Type of a structure representing an USB driver.
- `typedef uint8_t usbep_t`
Type of an endpoint identifier.
- `typedef void(* usbcallback_t)(USBDriver *usbp)`
Type of an USB generic notification callback.
- `typedef void(* usbepcallback_t)(USBDriver *usbp, usbep_t ep)`
Type of an USB endpoint callback.
- `typedef void(* usbeventcb_t)(USBDriver *usbp, usbevent_t event)`
Type of an USB event notification callback.
- `typedef bool_t(* usbreqhandler_t)(USBDriver *usbp)`
Type of a requests handler callback.
- `typedef const USBDescriptor *(* usbgetdescriptor_t)(USBDriver *usbp, uint8_t dtype, uint8_t dindex, uint16_t lang)`
Type of an USB descriptor-retrieving callback.

Enumerations

- `enum usbstate_t {`
`USB_UNINIT = 0, USB_STOP = 1, USB_READY = 2, USB_SELECTED = 3,`
`USB_ACTIVE = 4 }`
Type of a driver state machine possible states.
- `enum usbepstatus_t { EP_STATUS_DISABLED = 0, EP_STATUS_STALLED = 1, EP_STATUS_ACTIVE = 2 }`
Type of an endpoint status.
- `enum usbep0state_t {`
`USB_EP0_WAITING_SETUP, USB_EP0_TX, USB_EP0_WAITING_TX0, USB_EP0_WAITING_STS,`
`USB_EP0_RX, USB_EP0_SENDING_STS, USB_EP0_ERROR }`
Type of an endpoint zero state machine states.
- `enum usbevent_t {`
`USB_EVENT_RESET = 0, USB_EVENT_ADDRESS = 1, USB_EVENT_CONFIGURED = 2, USB_EVENT_SUSPEND = 3,`
`USB_EVENT_WAKEUP = 4, USB_EVENT_STALLED = 5 }`
Type of an enumeration of the possible USB events.

13.153 usb_lld.c File Reference

13.153.1 Detailed Description

USB Driver subsystem low level driver source template.

```
#include "ch.h"
#include "hal.h"
```

Functions

- void `usb_lld_init` (void)

Low level USB driver initialization.
- void `usb_lld_start` (USBDriver *usbp)

Configures and activates the USB peripheral.
- void `usb_lld_stop` (USBDriver *usbp)

Deactivates the USB peripheral.
- void `usb_lld_reset` (USBDriver *usbp)

USB low level reset routine.
- void `usb_lld_set_address` (USBDriver *usbp)

Sets the USB address.
- void `usb_lld_init_endpoint` (USBDriver *usbp, usbep_t ep)

Enables an endpoint.
- void `usb_lld_disable_endpoints` (USBDriver *usbp)

Disables all the active endpoints except the endpoint zero.
- usbepstatus_t `usb_lld_get_status_out` (USBDriver *usbp, usbep_t ep)

Returns the status of an OUT endpoint.
- usbepstatus_t `usb_lld_get_status_in` (USBDriver *usbp, usbep_t ep)

Returns the status of an IN endpoint.
- void `usb_lld_read_setup` (USBDriver *usbp, usbep_t ep, uint8_t *buf)

Reads a setup packet from the dedicated packet buffer.
- void `usb_lld_prepare_receive` (USBDriver *usbp, usbep_t ep)

Prepares for a receive operation.
- void `usb_lld_prepare_transmit` (USBDriver *usbp, usbep_t ep)

Prepares for a transmit operation.
- void `usb_lld_start_out` (USBDriver *usbp, usbep_t ep)

Starts a receive operation on an OUT endpoint.
- void `usb_lld_start_in` (USBDriver *usbp, usbep_t ep)

Starts a transmit operation on an IN endpoint.
- void `usb_lld_stall_out` (USBDriver *usbp, usbep_t ep)

Brings an OUT endpoint in the stalled state.
- void `usb_lld_stall_in` (USBDriver *usbp, usbep_t ep)

Brings an IN endpoint in the stalled state.
- void `usb_lld_clear_out` (USBDriver *usbp, usbep_t ep)

Brings an OUT endpoint in the active state.
- void `usb_lld_clear_in` (USBDriver *usbp, usbep_t ep)

Brings an IN endpoint in the active state.

Variables

- USBDriver `USB1`

USB1 driver identifier.

13.153.2 Variable Documentation

13.153.2.1 **USBInEndpointState** in

IN EP0 state.

13.153.2.2 **USBOutEndpointState** out

OUT EP0 state.

13.154 usb_ll.h File Reference

13.154.1 Detailed Description

USB Driver subsystem low level driver header template.

Data Structures

- struct **USBInEndpointState**
Type of an IN endpoint state structure.
- struct **USBOutEndpointState**
Type of an OUT endpoint state structure.
- struct **USBEndpointConfig**
Type of an USB endpoint configuration structure.
- struct **USBConfig**
Type of an USB driver configuration structure.
- struct **USBDriver**
Structure representing an USB driver.

Functions

- void **usb_ll_init** (void)
Low level USB driver initialization.
- void **usb_ll_start** (**USBDriver** *usbp)
Configures and activates the USB peripheral.
- void **usb_ll_stop** (**USBDriver** *usbp)
Deactivates the USB peripheral.
- void **usb_ll_reset** (**USBDriver** *usbp)
USB low level reset routine.
- void **usb_ll_set_address** (**USBDriver** *usbp)
Sets the USB address.
- void **usb_ll_init_endpoint** (**USBDriver** *usbp, **usbep_t** ep)
Enables an endpoint.
- void **usb_ll_disable_endpoints** (**USBDriver** *usbp)
Disables all the active endpoints except the endpoint zero.
- **usbepstatus_t** **usb_ll_get_status_in** (**USBDriver** *usbp, **usbep_t** ep)
Returns the status of an IN endpoint.
- **usbepstatus_t** **usb_ll_get_status_out** (**USBDriver** *usbp, **usbep_t** ep)
Returns the status of an OUT endpoint.
- void **usb_ll_read_setup** (**USBDriver** *usbp, **usbep_t** ep, **uint8_t** *buf)

- `void usb_ll_prepare_receive (USBDriver *usbp, usbep_t ep)`
Prepares for a receive operation.
- `void usb_ll_prepare_transmit (USBDriver *usbp, usbep_t ep)`
Prepares for a transmit operation.
- `void usb_ll_start_out (USBDriver *usbp, usbep_t ep)`
Starts a receive operation on an OUT endpoint.
- `void usb_ll_start_in (USBDriver *usbp, usbep_t ep)`
Starts a transmit operation on an IN endpoint.
- `void usb_ll_stall_out (USBDriver *usbp, usbep_t ep)`
Brings an OUT endpoint in the stalled state.
- `void usb_ll_stall_in (USBDriver *usbp, usbep_t ep)`
Brings an IN endpoint in the stalled state.
- `void usb_ll_clear_out (USBDriver *usbp, usbep_t ep)`
Brings an OUT endpoint in the active state.
- `void usb_ll_clear_in (USBDriver *usbp, usbep_t ep)`
Brings an IN endpoint in the active state.

Defines

- `#define USB_MAX_ENDPOINTS 4`
Maximum endpoint address.
- `#define USB_EP0_STATUS_STAGE USB_EP0_STATUS_STAGE_SW`
Status stage handling method.
- `#define USB_SET_ADDRESS_MODE USB_EARLY_SET_ADDRESS`
The address can be changed immediately upon packet reception.
- `#define USB_SET_ADDRESS_ACK_HANDLING USB_SET_ADDRESS_ACK_SW`
Method for set address acknowledge.
- `#define usb_ll_get_transaction_size(usbp, ep) ((usbp)->epc[ep]->out_state->rxcnt)`
Returns the exact size of a receive transaction.
- `#define usb_ll_connect_bus(usbp)`
Connects the USB device.
- `#define usb_ll_disconnect_bus(usbp)`
Disconnect the USB device.

Configuration options

- `#define PLATFORM_USB_USE_USB1 FALSE`
USB driver enable switch.

Index

_BSEMAPHORE_DATA
 Binary Semaphores, 101
_CHIBIOS_RT_
 Version Numbers and Identification, 38
_CONDVAR_DATA
 Condition Variables, 120
_EVENTSOURCE_DATA
 Event Flags, 131
_INPUTQUEUE_DATA
 I/O Queues, 159
_IOBUS_DATA
 PAL Driver, 334
_MAILBOX_DATA
 Mailboxes, 147
_MEMORYPOOL_DATA
 Memory Pools, 173
_MUTEX_DATA
 Mutexes, 112
_OUTPUTQUEUE_DATA
 I/O Queues, 161
_SEMAPHORE_DATA
 Counting Semaphores, 99
_THREADSQUEUE_DATA
 Internals, 199
_adc_isr_error_code
 ADC Driver, 219
_adc_isr_full_code
 ADC Driver, 219
_adc_isr_half_code
 ADC Driver, 219
_adc_reset_i
 ADC Driver, 217
_adc_reset_s
 ADC Driver, 217
_adc_timeout_isr
 ADC Driver, 218
_adc_wakeup_isr
 ADC Driver, 218
_base_asynchronous_channel_data
 Abstract I/O Channel, 296
_base_asynchronous_channel_methods
 Abstract I/O Channel, 296
_base_block_device_data
 Abstract I/O Block Device, 287
_base_block_device_methods
 Abstract I/O Block Device, 286
_base_channel_data
 Abstract I/O Channel, 293
_base_channel_methods
 Abstract I/O Channel, 293
_base_file_stream_data
 Abstract File Streams, 182
_base_file_stream_methods
 Abstract File Streams, 182
_base_sequential_stream_data
 Abstract Sequential Streams, 179
_base_sequential_stream_methods
 Abstract Sequential Streams, 179
_core_init
 Core Memory Manager, 164
_heap_init
 Heaps, 167
_icu_isr_invoke_overflow_cb
 ICU Driver, 283
_icu_isr_invoke_period_cb
 ICU Driver, 283
_icu_isr_invoke_width_cb
 ICU Driver, 283
_idle_thread
 System Management, 54
 Version Numbers and Identification, 38
_memory_stream_data
 Memory Streams, 480
_mmc_driver_methods
 MMC over SPI Driver, 321
_mmcsd_block_device_data
 MMC/SD Block Device, 326
_mmcsd_block_device_methods
 MMC/SD Block Device, 326
_pal_lld_init
 PAL Driver, 332
_pal_lld_setgroupmode
 PAL Driver, 332
_scheduler_init
 Scheduler, 61
_sdc_driver_methods
 SDC Driver, 380
_sdc_wait_for_transfer_state
 SDC Driver, 373
_serial_driver_data
 Serial Driver, 393
_serial_driver_methods
 Serial Driver, 389
_serial_usb_driver_data
 Serial over USB Driver, 400
_serial_usb_driver_methods
 Serial over USB Driver, 401
_spi_isr_code
 SPI Driver, 419
_spi_wait_s

SPI Driver, 418
_spi_wakeup_isr
 SPI Driver, 419
_thread_init
 Threads, 71
_thread_memfill
 Threads, 72
_trace_init
 Debug, 189
_usb_ep0in
 USB Driver, 453
_usb_ep0out
 USB Driver, 454
_usb_ep0setup
 USB Driver, 452
_usb_isr_invoke_event_cb
 USB Driver, 466
_usb_isr_invoke_in_cb
 USB Driver, 467
_usb_isr_invoke_out_cb
 USB Driver, 467
_usb_isr_invoke_setup_cb
 USB Driver, 466
_usb_isr_invoke_sof_cb
 USB Driver, 466
_usb_reset
 USB Driver, 452
_vt_init
 Time and Virtual Timers, 84

ABSPRIO
 Scheduler, 67

Abstract File Streams, 181
 _base_file_stream_data, 182
 _base_file_stream_methods, 182
 chFileStreamClose, 182
 chFileStreamGetError, 183
 chFileStreamGetPosition, 183
 chFileStreamGetSize, 183
 chFileStreamSeek, 183
 FILE_ERROR, 182
 FILE_OK, 182
 fileoffset_t, 184

Abstract I/O Block Device, 285
 _base_block_device_data, 287
 _base_block_device_methods, 286
 BLK_ACTIVE, 291
 BLK_CONNECTING, 291
 BLK_DISCONNECTING, 291
 BLK_READING, 291
 BLK_READY, 291
 BLK_STOP, 291
 BLK_SYNCING, 291
 BLK_UNINIT, 291
 BLK_WRITING, 291
 blkConnect, 288
 blkDisconnect, 289
 blkGetDriverState, 287
 blkGetInfo, 290

blkIsInserted, 288
blkIsTransferring, 287
blkIsWriteProtected, 288
blkRead, 289
blkstate_t, 291
blkSync, 290
blkWrite, 290

Abstract I/O Channel, 291
 _base_asynchronous_channel_data, 296
 _base_asynchronous_channel_methods, 296
 _base_channel_data, 293
 _base_channel_methods, 293
 CHN_CONNECTED, 296
 CHN_DISCONNECTED, 296
 CHN_INPUT_AVAILABLE, 296
 CHN_NO_ERROR, 295
 CHN_OUTPUT_EMPTY, 296
 CHN_TRANSMISSION_END, 296
 chnAddFlags, 296
 chnGetEventSource, 296
 chnGetTimeout, 293
 chnPutTimeout, 293
 chnRead, 295
 chnReadTimeout, 295
 chnWrite, 294
 chnWriteTimeout, 294

Abstract Sequential Streams, 178
 _base_sequential_stream_data, 179
 _base_sequential_stream_methods, 179
 chSequentialStreamGet, 180
 chSequentialStreamPut, 180
 chSequentialStreamRead, 180
 chSequentialStreamWrite, 179

ADC Driver, 205
 _adc_isr_error_code, 219
 _adc_isr_full_code, 219
 _adc_isr_half_code, 219
 adc_reset_i, 217
 adc_reset_s, 217
 adc_timeout_isr, 218
 adc_wakeup_isr, 218
 ADC_ACTIVE, 221
 ADC_COMPLETE, 221
 ADC_ERR_DMAFAILURE, 221
 ADC_ERR_OVERFLOW, 221
 ADC_ERROR, 221
 ADC_READY, 221
 ADC_STOP, 221
 ADC_UNINIT, 221
 adc_channels_num_t, 220
 adc_lld_init, 215
 adc_lld_start, 216
 adc_lld_start_conversion, 216
 adc_lld_stop, 216
 adc_lld_stop_conversion, 216
 ADC_USE_MUTUAL_EXCLUSION, 217
 ADC_USE_WAIT, 217
 adcAcquireBus, 213
 adccallback_t, 220

adcConvert, 214
ADCD1, 217
ADCDriver, 220
adcerror_t, 221
adcerrorcallback_t, 221
adclInit, 209
adcObjectInit, 209
adcReleaseBus, 214
adcsample_t, 220
adcStart, 210
adcStartConversion, 211
adcStartConversionl, 211
adcstate_t, 221
adcStop, 210
adcStopConversion, 212
adcStopConversionl, 212
PLATFORM_ADC_USE_ADC1, 220
adc.c, 736
adc.h, 737
ADC_ACTIVE
 ADC Driver, 221
ADC_COMPLETE
 ADC Driver, 221
ADC_ERR_DMAFAILURE
 ADC Driver, 221
ADC_ERR_OVERFLOW
 ADC Driver, 221
ADC_ERROR
 ADC Driver, 221
ADC_READY
 ADC Driver, 221
ADC_STOP
 ADC Driver, 221
ADC_UNINIT
 ADC Driver, 221
adc_channels_num_t
 ADC Driver, 220
adc_lld.c, 738
adc_lld.h, 738
adc_lld_init
 ADC Driver, 215
adc_lld_start
 ADC Driver, 216
adc_lld_start_conversion
 ADC Driver, 216
adc_lld_stop
 ADC Driver, 216
adc_lld_stop_conversion
 ADC Driver, 216
ADC_USE_MUTUAL_EXCLUSION
 ADC Driver, 217
 Configuration, 477
ADC_USE_WAIT
 ADC Driver, 217
 Configuration, 477
adcAcquireBus
 ADC Driver, 213
adccallback_t
 ADC Driver, 220
ADCConfig, 500
ADCConversionGroup, 500
 circular, 502
 end_cb, 502
 error_cb, 502
 num_channels, 502
adcConvert
 ADC Driver, 214
ADCD1
 ADC Driver, 217
ADCDriver, 502
 ADC Driver, 220
 config, 504
 depth, 504
 grpp, 504
 mutex, 504
 samples, 504
 state, 504
 thread, 504
adcerror_t
 ADC Driver, 221
adcerrorcallback_t
 ADC Driver, 221
adclInit
 ADC Driver, 209
adcObjectInit
 ADC Driver, 209
adcReleaseBus
 ADC Driver, 214
adcsample_t
 ADC Driver, 220
adcStart
 ADC Driver, 210
adcStartConversion
 ADC Driver, 211
adcStartConversionl
 ADC Driver, 211
adcstate_t
 ADC Driver, 221
adcStop
 ADC Driver, 210
adcStopConversion
 ADC Driver, 212
adcStopConversionl
 ADC Driver, 212
addCounterl
 chibios_rt::CounterSemaphore, 571
addEvents
 chibios_rt::BaseThread, 532
address
 USBDriver, 724
ALL_EVENTS
 Event Flags, 132
alloc
 chibios_rt::Core, 563
 chibios_rt::MemoryPool, 628
allocl
 chibios_rt::Core, 563
 chibios_rt::MemoryPool, 628

Base Kernel Services, 51
BaseAsynchronousChannel, 504
 vmt, 506
BaseAsynchronousChannelVMT, 506
BaseBlockDevice, 508
 vmt, 510
BaseBlockDeviceVMT, 510
BaseChannel, 510
 vmt, 512
BaseChannelVMT, 512
BaseFileStream, 514
 vmt, 515
BaseFileStreamVMT, 515
BaseSequentialStream, 516
 vmt, 518
BaseSequentialStreamVMT, 520
BaseStaticThread
 chibios_rt::BaseStaticThread, 523
BaseThread
 chibios_rt::BaseThread, 527
best
 TimeMeasurement, 713
Binary Semaphores, 100
 _BSEMAPHORE_DATA, 101
 BSEMAPHORE_DECL, 101
 chBSemGetStatel, 105
 chBSemInit, 102
 chBSemReset, 103
 chBSemResetl, 104
 chBSemSignal, 104
 chBSemSignall, 104
 chBSemWait, 102
 chBSemWaitS, 102
 chBSemWaitTimeout, 102
 chBSemWaitTimeoutS, 103
BinarySemaphore, 538
 chibios_rt::BinarySemaphore, 541
BLK_ACTIVE
 Abstract I/O Block Device, 291
BLK_CONNECTING
 Abstract I/O Block Device, 291
BLK_DISCONNECTING
 Abstract I/O Block Device, 291
BLK_READING
 Abstract I/O Block Device, 291
BLK_READY
 Abstract I/O Block Device, 291
BLK_STOP
 Abstract I/O Block Device, 291
BLK_SYNCING
 Abstract I/O Block Device, 291
BLK_UNINIT
 Abstract I/O Block Device, 291
BLK_WRITING
 Abstract I/O Block Device, 291
blk_num
 BlockDeviceInfo, 544
blk_size
 BlockDeviceInfo, 544
blkConnect
 Abstract I/O Block Device, 288
blkDisconnect
 Abstract I/O Block Device, 289
blkGetDriverState
 Abstract I/O Block Device, 287
blkGetInfo
 Abstract I/O Block Device, 290
blkIsInserted
 Abstract I/O Block Device, 288
blkIsTransferring
 Abstract I/O Block Device, 287
blkIsWriteProtected
 Abstract I/O Block Device, 288
blkRead
 Abstract I/O Block Device, 289
blkstate_t
 Abstract I/O Block Device, 291
blkSync
 Abstract I/O Block Device, 290
blkWrite
 Abstract I/O Block Device, 290
BlockDeviceInfo, 544
 blk_num, 544
 blk_size, 544
bool_t
 Types, 50
broadcast
 chibios_rt::CondVar, 559
broadcastFlags
 chibios_rt::EvtSource, 580
broadcastFlagsl
 chibios_rt::EvtSource, 580
broadcastl
 chibios_rt::CondVar, 559
bsem
 chibios_rt::BinarySemaphore, 544
BSEMAPHORE_DECL
 Binary Semaphores, 101
bulk_in
 SerialUSBConfig, 677
bulk_out
 SerialUSBConfig, 677
C++ Wrapper, 479
c_queue
 CondVar, 556
callback
 GPTConfig, 588
 PWMChannelConfig, 660
 PWMConfig, 662
CAN Driver, 221
 CAN_READY, 233
 CAN_SLEEP, 233
 CAN_STARTING, 233
 CAN_STOP, 233
 CAN_UNINIT, 233
 CAN_ANY_MAILBOX, 232
 CAN_BUS_OFF_ERROR, 232

CAN_FRAMING_ERROR, 232
CAN_LIMIT_ERROR, 232
CAN_LIMIT_WARNING, 232
can_lld_init, 229
can_lld_is_rx_nonempty, 231
can_lld_is_tx_empty, 230
can_lld_receive, 231
can_lld_sleep, 231
can_lld_start, 230
can_lld_stop, 230
can_lld_transmit, 230
can_lld_wakeup, 232
CAN_MAILBOX_TO_MASK, 233
CAN_OVERFLOW_ERROR, 232
CAN_RX_MAILBOXES, 233
CAN_SUPPORTS_SLEEP, 233
CAN_TX_MAILBOXES, 233
CAN_USE_SLEEP_MODE, 232
CAND1, 232
canInit, 224
canmbx_t, 233
canObjectInit, 225
canReceive, 227
canSleep, 228
canStart, 225
canstate_t, 233
canStop, 226
canTransmit, 226
canWakeUp, 229
PLATFORM_CAN_USE_CAN1, 233
can.c, 739
can.h, 740
CAN_READY
 CAN Driver, 233
CAN_SLEEP
 CAN Driver, 233
CAN_STARTING
 CAN Driver, 233
CAN_STOP
 CAN Driver, 233
CAN_UNINIT
 CAN Driver, 233
CAN_ANY_MAILBOX
 CAN Driver, 232
CAN_BUS_OFF_ERROR
 CAN Driver, 232
CAN_FRAMING_ERROR
 CAN Driver, 232
CAN_LIMIT_ERROR
 CAN Driver, 232
CAN_LIMIT_WARNING
 CAN Driver, 232
can_lld.c, 741
can_lld.h, 742
can_lld_init
 CAN Driver, 229
can_lld_is_rx_nonempty
 CAN Driver, 231
can_lld_is_tx_empty
 CAN Driver, 230
can_lld_receive
 CAN Driver, 231
can_lld_sleep
 CAN Driver, 231
can_lld_start
 CAN Driver, 230
can_lld_stop
 CAN Driver, 230
can_lld_transmit
 CAN Driver, 230
can_lld_wakeup
 CAN Driver, 232
CAN_MAILBOX_TO_MASK
 CAN Driver, 233
CAN_OVERFLOW_ERROR
 CAN Driver, 232
CAN_RX_MAILBOXES
 CAN Driver, 233
CAN_SUPPORTS_SLEEP
 CAN Driver, 233
CAN_TX_MAILBOXES
 CAN Driver, 233
CAN_USE_SLEEP_MODE
 CAN Driver, 232
 Configuration, 477
CANConfig, 544
CAND1
 CAN Driver, 232
CANDriver, 545
 config, 546
 error_event, 547
 rxfull_event, 546
 rxsem, 546
 sleep_event, 547
 state, 546
 txempty_event, 546
 txsem, 546
 wakeup_event, 547
CANFilter, 547
canInit
 CAN Driver, 224
canmbx_t
 CAN Driver, 233
canObjectInit
 CAN Driver, 225
canReceive
 CAN Driver, 227
CANRxFrame, 547
 data16, 548
 data32, 548
 data8, 548
 DLC, 548
 EID, 548
 IDE, 548
 RTR, 548
 SID, 548
canSleep
 CAN Driver, 228

canStart
 CAN Driver, 225
canstate_t
 CAN Driver, 233
canStop
 CAN Driver, 226
canTransmit
 CAN Driver, 226
CANTxFrame, 548
 data16, 549
 data32, 549
 data8, 549
 DLC, 548
 EID, 549
 IDE, 549
 RTR, 548
 SID, 549
canWakeup
 CAN Driver, 229
cardmode
 SDCDriver, 668
cb
 EXTChannelConfig, 582
cdc_linecoding_t, 549
cf_off_ctxt
 chdebug_t, 554
cf_off_flags
 chdebug_t, 555
cf_off_name
 chdebug_t, 555
cf_off_newer
 chdebug_t, 555
cf_off_older
 chdebug_t, 555
cf_off_preempt
 chdebug_t, 555
cf_off_prio
 chdebug_t, 554
cf_off_refs
 chdebug_t, 555
cf_off_state
 chdebug_t, 555
cf_off_stklimit
 chdebug_t, 555
cf_off_time
 chdebug_t, 555
ch.cpp, 743
ch.h, 743
ch.hpp, 745
CH_ARCHITECTURE_NAME
 Port, 203
CH_ARCHITECTURE_VARIANT_NAME
 Port, 203
CH_ARCHITECTURE_XXX
 Port, 203
CH_COMPILER_NAME
 Port, 204
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_SYSTEM_STATE_CHECK
 Configuration, 46
CH_DBG_THREADS_PROFILING
 Configuration, 47
CH_FAILED
 Version Numbers and Identification, 39
CH_FAST_IRQ_HANDLER
 System Management, 58
CH_FREQUENCY
 Configuration, 41
ch_identifier
 chdebug_t, 554
CH_IRQ_EPILOGUE
 System Management, 58
CH_IRQ_HANDLER
 System Management, 58
CH_IRQ_PROLOGUE
 System Management, 58
CH_KERNEL_MAJOR
 Version Numbers and Identification, 39
CH_KERNEL_MINOR
 Version Numbers and Identification, 39
CH_KERNEL_PATCH
 Version Numbers and Identification, 39
CH_KERNEL_VERSION
 Version Numbers and Identification, 38
CH_MEMCORE_SIZE
 Configuration, 41
CH_NO_IDLE_THREAD
 Configuration, 42
CH_OPTIMIZE_SPEED
 Configuration, 42
CH_PORT_INFO
 Port, 204
ch_ptrsize
 chdebug_t, 554
ch_size
 chdebug_t, 554
CH_STACK_FILL_VALUE
 Debug, 194
CH_SUCCESS
 Version Numbers and Identification, 39
ch_swc_event_t, 549
 se_state, 551
 se_time, 551
 se_tp, 551
 se_wtobjp, 551
CH_THREAD_FILL_VALUE
 Debug, 194
ch_threadsize
 chdebug_t, 554

CH_TIME_QUANTUM
 Configuration, 41

ch_timesize
 chdebug_t, 554

CH_TRACE_BUFFER_SIZE
 Debug, 194

ch_trace_buffer_t, 551
 tb_buffer, 553
 tb_ptr, 553
 tb_size, 553

CH_USE_CONDVARSL
 Configuration, 43

CH_USE_CONDVARSL_TIMEOUT
 Configuration, 43

CH_USE_DYNAMIC
 Configuration, 46

CH_USE_EVENTS
 Configuration, 44

CH_USE_EVENTS_TIMEOUT
 Configuration, 44

CH_USE_HEAP
 Configuration, 45

CH_USE_MAILBOXES
 Configuration, 44

CH_USE_MALLOC_HEAP
 Configuration, 45

CH_USE_MEMCORE
 Configuration, 45

CH_USE_MEMPOOLS
 Configuration, 45

CH_USE_MESSAGES
 Configuration, 44

CH_USE_MESSAGES_PRIORITY
 Configuration, 44

CH_USE_MUTEXES
 Configuration, 43

CH_USE_QUEUES
 Configuration, 45

CH_USE_REGISTRY
 Configuration, 42

CH_USE_SEMAPHORES
 Configuration, 42

CH_USE_SEMAPHORES_PRIORITY
 Configuration, 43

CH_USE_SEMSW
 Configuration, 43

CH_USE_WAITEXIT
 Configuration, 42

ch_version
 chdebug_t, 554

ch_zero
 chdebug_t, 554

channels
 EXTConfig, 583
 PWMConfig, 662

chbsem.h, 746

chBSemGetStatel
 Binary Semaphores, 105

chBSemInit

 Binary Semaphores, 102

chBSemReset
 Binary Semaphores, 103

chBSemResetl
 Binary Semaphores, 104

chBSemSignal
 Binary Semaphores, 104

chBSemSignall
 Binary Semaphores, 104

chBSemWait
 Binary Semaphores, 102

chBSemWaitS
 Binary Semaphores, 102

chBSemWaitTimeout
 Binary Semaphores, 102

chBSemWaitTimeoutS
 Binary Semaphores, 103

chcond.c, 746

chcond.h, 747

chCondBroadcast
 Condition Variables, 115

chCondBroadcastl
 Condition Variables, 116

chCondInit
 Condition Variables, 114

chCondSignal
 Condition Variables, 114

chCondSignall
 Condition Variables, 114

chCondWait
 Condition Variables, 116

chCondWaitS
 Condition Variables, 117

chCondWaitTimeout
 Condition Variables, 118

chCondWaitTimeoutS
 Condition Variables, 119

chconf.h, 748

chcore.c, 750

chcore.h, 750

chCoreAlloc
 Core Memory Manager, 164

chCoreAllocl
 Core Memory Manager, 165

chCoreStatus
 Core Memory Manager, 165

chDbgAssert
 Debug, 195

chDbgCheck
 Debug, 195

chDbgCheckClassl
 Debug, 193

chDbgCheckClassS
 Debug, 193

chDbgPanic
 Debug, 194

chdebug.c, 752

chdebug.h, 753

chdebug_t, 553

cf_off_ctxt, 554
cf_off_flags, 555
cf_off_name, 555
cf_off_newer, 555
cf_off_older, 555
cf_off_preempt, 555
cf_off_prio, 554
cf_off_refs, 555
cf_off_state, 555
cf_off_stklimit, 555
cf_off_time, 555
ch_identifier, 554
ch_ptrsize, 554
ch_size, 554
ch_threadsize, 554
ch_timesize, 554
ch_version, 554
ch_zero, 554
chdynamic.c, 754
chdynamic.h, 754
chevents.c, 754
chevents.h, 755
chEvtAddEvents
 Event Flags, 124
chEvtBroadcast
 Event Flags, 133
chEvtBroadcastFlags
 Event Flags, 125
chEvtBroadcastFlagsI
 Event Flags, 126
chEvtBroadcastI
 Event Flags, 133
chEvtDispatch
 Event Flags, 127
chEvtGetAndClearEvents
 Event Flags, 123
chEvtGetAndClearFlags
 Event Flags, 124
chEvtGetAndClearFlagsI
 Event Flags, 124
chEvtInit
 Event Flags, 132
chEvtIsListeningI
 Event Flags, 133
chEvtRegister
 Event Flags, 132
chEvtRegisterMask
 Event Flags, 123
chEvtSignal
 Event Flags, 124
chEvtSignall
 Event Flags, 125
chEvtUnregister
 Event Flags, 123
chEvtWaitAll
 Event Flags, 131
chEvtWaitAllTimeout
 Event Flags, 129
chEvtWaitAny
 Event Flags, 130
chEvtWaitAnyTimeout
 Event Flags, 128
chEvtWaitOne
 Event Flags, 129
chEvtWaitOneTimeout
 Event Flags, 127
chfiles.h, 757
chFileStreamClose
 Abstract File Streams, 182
chFileStreamGetError
 Abstract File Streams, 183
chFileStreamGetPosition
 Abstract File Streams, 183
chFileStreamGetSize
 Abstract File Streams, 183
chFileStreamSeek
 Abstract File Streams, 183
chheap.c, 757
chheap.h, 758
chHeapAlloc
 Heaps, 168
chHeapFree
 Heaps, 168
chHeapInit
 Heaps, 167
chHeapStatus
 Heaps, 169
chibios_rt, 498
chibios_rt::BaseSequentialStreamInterface, 518
 get, 519
 put, 519
 read, 518
 write, 518
chibios_rt::BaseStaticThread, 520
 BaseStaticThread, 523
 start, 523
chibios_rt::BaseThread, 523
 addEvents, 532
 BaseThread, 527
 dispatchEvents, 536
 exit, 528
 exitS, 529
 getAndClearEvents, 531
 main, 527
 setName, 527
 setPriority, 527
 shouldTerminate, 529
 sleep, 530
 sleepUntil, 530
 start, 527
 unlockAllMutexes, 538
 unlockMutex, 537
 unlockMutexS, 537
 waitForAllEvents, 534
 waitForAllEventsTimeout, 535
 waitForAnyEvent, 533
 waitForAnyEventTimeout, 535
 waitForMessage, 531

waitOneEvent, 532
waitOneEventTimeout, 534
yield, 531
chibios_rt::BinarySemaphore, 540
 BinarySemaphore, 541
 bsem, 544
 getStatel, 543
 reset, 542
 resetl, 543
 signal, 543
 signall, 543
 wait, 541
 waitS, 541
 waitForTimeOut, 541
 waitForTimeOutS, 542
chibios_rt::CondVar, 557
 broadcast, 559
 broadcastl, 559
 CondVar, 558
 condvar, 562
 signal, 558
 signall, 558
 wait, 560
 waitS, 560
 waitForTimeOut, 561
chibios_rt::Core, 562
 alloc, 563
 allocI, 563
 getStatus, 564
chibios_rt::CounterSemaphore, 564
 addCounterl, 571
 CounterSemaphore, 566
 getCounterl, 571
 reset, 566
 resetl, 567
 sem, 572
 signal, 570
 signall, 570
 signalWait, 571
 wait, 568
 waitS, 568
 waitForTimeOut, 569
 waitForTimeOutS, 569
chibios_rt::EvtListener, 576
 ev_listener, 578
 getAndClearFlags, 577
 getAndClearFlagsI, 577
chibios_rt::EvtSource, 578
 broadcastFlags, 580
 broadcastFlagsI, 580
 ev_source, 581
 EvtSource, 578
 registerMask, 579
 registerOne, 579
 unregister, 579
chibios_rt::InQueue, 596
 get, 601
 getEmptyI, 599
 getFullI, 598
 getTimeout, 601
 InQueue, 598
 isEmptyI, 599
 isFullI, 599
 putI, 600
 readTimeout, 602
 resetl, 599
chibios_rt::InQueueBuffer, 603
 InQueueBuffer, 604
chibios_rt::Mailbox, 609
 fetch, 615
 fetchI, 617
 fetchS, 616
 getFreeCountI, 617
 getUsedCountI, 618
 Mailbox, 611
 mb, 618
 post, 611
 postAhead, 613
 postAheadI, 615
 postAheadS, 614
 postI, 613
 postS, 612
 reset, 611
chibios_rt::MailboxBuffer, 620
 MailboxBuffer, 622
chibios_rt::MemoryPool, 625
 alloc, 628
 allocI, 628
 free, 629
 freeI, 630
 loadArray, 627
 MemoryPool, 626, 627
 pool, 630
chibios_rt::Mutex, 643
 lock, 646
 lockS, 646
 Mutex, 644
 mutex, 647
 tryLock, 644
 tryLockS, 645
chibios_rt::ObjectsPool, 647
 ObjectsPool, 649
chibios_rt::OutQueue, 649
 getEmptyI, 652
 getFullI, 651
 getI, 654
 isEmptyI, 652
 isFullI, 652
 OutQueue, 651
 put, 653
 putTimeout, 653
 resetI, 652
 writeTimeout, 655
chibios_rt::OutQueueBuffer, 655
 OutQueueBuffer, 657
chibios_rt::System, 687
 getTime, 689
 init, 688

isTimeWithin, 690
lock, 688
lockFromIlsr, 689
unlock, 689
unlockFromIlsr, 689
chibios_rt::ThreadReference, 697
 getMessage, 705
 isPendingMessage, 705
 releaseMessage, 705
 requestTerminate, 703
 resume, 702
 resumel, 702
 sendMessage, 704
 signalEvents, 706
 signalEventsl, 706
 stop, 700
 suspend, 701
 suspendS, 701
 thread_ref, 707
 ThreadReference, 700
 wait, 703
chibios_rt::Timer, 714
 isArmedl, 715
 resetl, 715
 setl, 715
 timer_ref, 716
chinline.h, 758
chlIQGet
 I/O Queues, 158
chlIQGetEmptyl
 I/O Queues, 157
chlIQGetFulll
 I/O Queues, 157
chlIQGetTimeout
 I/O Queues, 152
chlIQInit
 I/O Queues, 150
chlIQIsEmptyl
 I/O Queues, 158
chlIQIsFulll
 I/O Queues, 158
chlIQPutl
 I/O Queues, 151
chlIQReadTimeout
 I/O Queues, 152
chlIQResetl
 I/O Queues, 150
chllists.c, 758
chllists.h, 759
chMBFetch
 Mailboxes, 144
chMBFetchl
 Mailboxes, 145
chMBFetchS
 Mailboxes, 144
chMBGetFreeCountl
 Mailboxes, 146
chMBGetUsedCountl
 Mailboxes, 147
chMBInit
 Mailboxes, 138
chmboxes.c, 760
chmboxes.h, 760
chMBPeekl
 Mailboxes, 147
chMBPost
 Mailboxes, 139
chMBPostAhead
 Mailboxes, 141
chMBPostAheadl
 Mailboxes, 143
chMBPostAheadS
 Mailboxes, 142
chMBPostl
 Mailboxes, 141
chMBPostS
 Mailboxes, 140
chMBReset
 Mailboxes, 139
chMBSizel
 Mailboxes, 146
chmemcore.c, 761
chmemcore.h, 762
chmempools.c, 763
chmempools.h, 763
chmsg.c, 764
chmsg.h, 764
chMsgGet
 Synchronous Messages, 136
chMsgIsPendingl
 Synchronous Messages, 136
chMsgRelease
 Synchronous Messages, 136
chMsgReleaseS
 Synchronous Messages, 136
chMsgSend
 Synchronous Messages, 134
chMsgWait
 Synchronous Messages, 135
chmtx.c, 765
chmtx.h, 765
chMtxInit
 Mutexes, 107
chMtxLock
 Mutexes, 107
chMtxLockS
 Mutexes, 108
chMtxQueueNotEmptyS
 Mutexes, 112
chMtxTryLock
 Mutexes, 108
chMtxTryLockS
 Mutexes, 109
chMtxUnlock
 Mutexes, 110
chMtxUnlockAll
 Mutexes, 111
chMtxUnlockS

Mutexes, 111
CHN_CONNECTED
 Abstract I/O Channel, 296
CHN_DISCONNECTED
 Abstract I/O Channel, 296
CHN_INPUT_AVAILABLE
 Abstract I/O Channel, 296
CHN_NO_ERROR
 Abstract I/O Channel, 295
CHN_OUTPUT_EMPTY
 Abstract I/O Channel, 296
CHN_TRANSMISSION_END
 Abstract I/O Channel, 296
chnAddFlags!
 Abstract I/O Channel, 296
chnGetEventSource
 Abstract I/O Channel, 296
chnGetTimeout
 Abstract I/O Channel, 293
chnPutTimeout
 Abstract I/O Channel, 293
chnRead
 Abstract I/O Channel, 295
chnReadTimeout
 Abstract I/O Channel, 295
chnWrite
 Abstract I/O Channel, 294
chnWriteTimeout
 Abstract I/O Channel, 294
chOQGetEmpty
 I/O Queues, 160
chOQGetFull
 I/O Queues, 159
chOQGetl
 I/O Queues, 154
chOQInit
 I/O Queues, 153
chOQIsEmpty
 I/O Queues, 160
chOQIsFull
 I/O Queues, 161
chOQPut
 I/O Queues, 161
chOQPutTimeout
 I/O Queues, 154
chOQResetl
 I/O Queues, 153
chOQWriteTimeout
 I/O Queues, 155
chPoolAdd
 Memory Pools, 174
chPoolAddl
 Memory Pools, 174
chPoolAlloc
 Memory Pools, 171
chPoolAllocl
 Memory Pools, 171
chPoolFree
 Memory Pools, 173
chPoolFreeI
 Memory Pools, 172
chPoolInit
 Memory Pools, 170
chPoolLoadArray
 Memory Pools, 170
chprintf.c, 766
chprintf.h, 767
CHPRINTF_USE_FLOAT
 System formatted print, 491
chQGetLink
 I/O Queues, 157
chQSizeI
 I/O Queues, 156
chQSpaceI
 I/O Queues, 156
chqueues.c, 767
chqueues.h, 768
chRegFirstThread
 Registry, 185
chRegGetThreadName
 Registry, 186
chregistry.c, 769
chregistry.h, 770
chRegNextThread
 Registry, 185
chRegSetThreadName
 Registry, 186
chrtclib.c, 770
chrtclib.h, 771
chSchCanYieldS
 Scheduler, 68
chsched.c, 771
chsched.h, 772
chSchDoReschedule
 Scheduler, 65
chSchDoRescheduleAhead
 Scheduler, 65
chSchDoRescheduleBehind
 Scheduler, 64
chSchDoYieldS
 Scheduler, 68
chSchGoSleepS
 Scheduler, 61
chSchGoSleepTimeoutS
 Scheduler, 62
chSchIsPreemptionRequired
 Scheduler, 64
chSchIsRescRequired
 Scheduler, 67
chSchPreemption
 Scheduler, 68
chSchReadyI
 Scheduler, 61
chSchRescheduleS
 Scheduler, 63
chSchWakeups
 Scheduler, 63
chsem.c, 774

chsem.h, 774
chSemAddCounterl
 Counting Semaphores, 97
chSemFastSignall
 Counting Semaphores, 99
chSemFastWaitl
 Counting Semaphores, 99
chSemGetCounterl
 Counting Semaphores, 100
chSemInit
 Counting Semaphores, 92
chSemReset
 Counting Semaphores, 92
chSemResetl
 Counting Semaphores, 93
chSemSignal
 Counting Semaphores, 96
chSemSignall
 Counting Semaphores, 97
chSemSignalWait
 Counting Semaphores, 98
chSemWait
 Counting Semaphores, 94
chSemWaitS
 Counting Semaphores, 94
chSemWaitTimeout
 Counting Semaphores, 95
chSemWaitTimeoutS
 Counting Semaphores, 95
chSequentialStreamGet
 Abstract Sequential Streams, 180
chSequentialStreamPut
 Abstract Sequential Streams, 180
chSequentialStreamRead
 Abstract Sequential Streams, 180
chSequentialStreamWriter
 Abstract Sequential Streams, 179
chsnprintf
 System formatted print, 490
chstreams.h, 775
chsys.c, 776
chsys.h, 776
chSysDisable
 System Management, 55
chSysEnable
 System Management, 56
chSysGetIdleThread
 System Management, 54
chSysHalt
 System Management, 55
chSysInit
 System Management, 53
chSysLock
 System Management, 56
chSysLockFromIsr
 System Management, 57
chSysSuspend
 System Management, 56
chSysSwitch

System Management, 55
chSysTimerHandlerl
 System Management, 53
chSysUnlock
 System Management, 57
chSysUnlockFromIsr
 System Management, 57
chThdAddRef
 Dynamic Threads, 175
chThdCreateFromHeap
 Dynamic Threads, 176
chThdCreateFromMemoryPool
 Dynamic Threads, 177
chThdCreate
 Threads, 72
chThdCreateStatic
 Threads, 73
chThdExit
 Threads, 76
chThdExitS
 Threads, 76
chThdGetPriority
 Threads, 80
chThdGetTicks
 Threads, 80
chThdLS
 Threads, 81
chThdRelease
 Dynamic Threads, 175
chThdResume
 Threads, 74
chThdResumel
 Threads, 82
chThdSelf
 Threads, 80
chThdSetPriority
 Threads, 73
chThdShouldTerminate
 Threads, 81
chThdSleep
 Threads, 75
chThdSleepMicroseconds
 Threads, 83
chThdSleepMilliseconds
 Threads, 82
chThdSleepS
 Threads, 82
chThdSleepSeconds
 Threads, 82
chThdSleepUntil
 Threads, 75
chThdTerminate
 Threads, 75
chThdTerminated
 Threads, 81
chThdWait
 Threads, 77
chThdYield
 Threads, 76

chthreads.c, 777
chthreads.h, 778
chTimeElapsedSince
 Time and Virtual Timers, 89
chTimelsWithin
 Time and Virtual Timers, 89
chTimeNow
 Time and Virtual Timers, 89
chtypes.h, 780
chvprintf
 System formatted print, 490
chvt.c, 781
chvt.h, 781
chVTDoTickI
 Time and Virtual Timers, 87
chVTIIsArmedI
 Time and Virtual Timers, 88
chVTRestet
 Time and Virtual Timers, 88
chVTRestetI
 Time and Virtual Timers, 85
chVTSet
 Time and Virtual Timers, 88
chVTSetI
 Time and Virtual Timers, 84
circular
 ADCConversionGroup, 502
cnt_t
 Types, 51
Command Shell, 482
 SHELL_MAX_ARGUMENTS, 485
 SHELL_MAX_LINE_LENGTH, 485
 shell_terminated, 485
 shellcmd_t, 486
 shellCreate, 483
 shellCreateStatic, 484
 shellExit, 483
 shellGetLine, 485
 shellInit, 483
Condition Variables, 113
 _CONDVAR_DATA, 120
 chCondBroadcast, 115
 chCondBroadcastI, 116
 chCondInit, 114
 chCondSignal, 114
 chCondSignall, 114
 chCondWait, 116
 chCondWaitS, 117
 chCondWaitTimeout, 118
 chCondWaitTimeoutS, 119
 CondVar, 120
 CONDVAR_DECL, 120
CondVar, 555
 c_queue, 556
 chibios_rt::CondVar, 558
 Condition Variables, 120
condvar
 chibios_rt::CondVar, 562
CONDVAR_DECL
Condition Variables, 120
config
 ADCDriver, 504
 CANDriver, 546
 EXTDriver, 584
 GPTDriver, 589
 I2CDriver, 593
 ICUDriver, 596
 MACDriver, 607
 MMCDriver, 636
 PWMDriver, 663
 SDCDriver, 668
 SPIDriver, 687
 UARTDriver, 719
 USBDriver, 723
Configuration, 39, 473
 ADC_USE_MUTUAL_EXCLUSION, 477
 ADC_USE_WAIT, 477
 CAN_USE_SLEEP_MODE, 477
 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_SYSTEM_STATE_CHECK, 46
 CH_DBG_THREADS_PROFILING, 47
 CH_FREQUENCY, 41
 CH_MEMCORE_SIZE, 41
 CH_NO_IDLE_THREAD, 42
 CH_OPTIMIZE_SPEED, 42
 CH_TIME_QUANTUM, 41
 CH_USE_CONDVARs, 43
 CH_USE_CONDVARs_TIMEOUT, 43
 CH_USE_DYNAMIC, 46
 CH_USE_EVENTS, 44
 CH_USE_EVENTS_TIMEOUT, 44
 CH_USE_HEAP, 45
 CH_USE_MAILBOXES, 44
 CH_USE_MALLOC_HEAP, 45
 CH_USE_MEMCORE, 45
 CH_USE_MEMPOOLS, 45
 CH_USE_MESSAGES, 44
 CH_USE_MESSAGES_PRIORITY, 44
 CH_USE_MUTEXES, 43
 CH_USE_QUEUES, 45
 CH_USE_REGISTRY, 42
 CH_USE_SEMAPHORES, 42
 CH_USE_SEMAPHORES_PRIORITY, 43
 CH_USE_SEMSW, 43
 CH_USE_WAITEXIT, 42
 HAL_USE_ADC, 475
 HAL_USE_CAN, 475
 HAL_USE_EXT, 475
 HAL_USE_GPT, 475
 HAL_USE_I2C, 476
 HAL_USE_ICU, 476
 HAL_USE_MAC, 476
 HAL_USE_MMCSPI, 476
 HAL_USE_PAL, 475

HAL_USE_PWM, 476
HAL_USE_RTC, 476
HAL_USE_SDC, 476
HAL_USE_SERIAL, 476
HAL_USE_SERIAL_USB, 476
HAL_USE_SPI, 476
HAL_USE_TM, 475
HAL_USE_UART, 476
HAL_USE_USB, 476
I2C_USE_MUTUAL_EXCLUSION, 477
IDLE_LOOP_HOOK, 48
MAC_USE_EVENTS, 477
MAC_USE_ZERO_COPY, 477
MMC_NICE_WAITING, 477
SDC_INIT_RETRY, 477
SDC_MMCSUPPORT, 477
SDC_NICE_WAITING, 478
SERIAL_BUFFERS_SIZE, 478
SERIAL_DEFAULT_BITRATE, 478
SERIAL_USB_BUFFERS_SIZE, 478
SPI_USE_MUTUAL_EXCLUSION, 478
SPI_USE_WAIT, 478
SYSTEM_HALT_HOOK, 48
SYSTEM_TICK_EVENT_HOOK, 48
THREAD_CONTEXT_SWITCH_HOOK, 48
THREAD_EXT_EXIT_HOOK, 48
THREAD_EXT_FIELDS, 47
THREAD_EXT_INIT_HOOK, 47
configuration
 USBDriver, 724
context, 562
Core Memory Manager, 163
 _core_init, 164
 chCoreAlloc, 164
 chCoreAllocI, 165
 chCoreStatus, 165
 MEM_ALIGN_MASK, 166
 MEM_ALIGN_NEXT, 166
 MEM_ALIGN_PREV, 166
 MEM_ALIGN_SIZE, 166
 MEM_IS_ALIGNED, 166
 memgetfunc_t, 166
CounterSemaphore
 chibios_rt::CounterSemaphore, 566
Counting Semaphores, 90
 _SEMAPHORE_DATA, 99
 chSemAddCounterI, 97
 chSemFastSignall, 99
 chSemFastWaitI, 99
 chSemGetCounterI, 100
 chSemInit, 92
 chSemReset, 92
 chSemResetI, 93
 chSemSignal, 96
 chSemSignall, 97
 chSemSignalWait, 98
 chSemWait, 94
 chSemWaitS, 94
 chSemWaitTimeout, 95
 chSemWaitTimeoutS, 95
 Semaphore, 100
 SEMAPHORE_DECL, 99
currp
 Scheduler, 67
data16
 CANRxFrame, 548
 CANTxFrame, 549
data32
 CANRxFrame, 548
 CANTxFrame, 549
data8
 CANRxFrame, 548
 CANTxFrame, 549
dbg_check_disable
 Debug, 189
dbg_check_enable
 Debug, 190
dbg_check_enter_isr
 Debug, 192
dbg_check_leave_isr
 Debug, 193
dbg_check_lock
 Debug, 190
dbg_check_lock_from_isr
 Debug, 191
dbg_check_suspend
 Debug, 190
dbg_check_unlock
 Debug, 191
dbg_check_unlock_from_isr
 Debug, 192
dbg_isr_cnt
 Debug, 194
dbg_lock_cnt
 Debug, 194
dbg_panic_msg
 Debug, 194
dbg_trace
 Debug, 189
dbg_trace_buffer
 Debug, 194
Debug, 187
 _trace_init, 189
 CH_STACK_FILL_VALUE, 194
 CH_THREAD_FILL_VALUE, 194
 CH_TRACE_BUFFER_SIZE, 194
 chDbgAssert, 195
 chDbgCheck, 195
 chDbgCheckClassI, 193
 chDbgCheckClassS, 193
 chDbgPanic, 194
 dbg_check_disable, 189
 dbg_check_enable, 190
 dbg_check_enter_isr, 192
 dbg_check_leave_isr, 193
 dbg_check_lock, 190
 dbg_check_lock_from_isr, 191

dbg_check_suspend, 190
dbg_check_unlock, 191
dbg_check_unlock_from_isr, 192
dbg_isr_cnt, 194
dbg_lock_cnt, 194
dbg_panic_msg, 194
dbg_trace, 189
dbg_trace_buffer, 194
DELAY_BETWEEN_TESTS
 Test Runtime, 496
depth
 ADCDriver, 504
dequeue
 Internals, 198
dispatchEvents
 chibios_rt::BaseThread, 536
DLC
 CANRxFrame, 548
 CANTxFrame, 548
Dynamic Threads, 175
 chThdAddRef, 175
 chThdCreateFromHeap, 176
 chThdCreateFromMemoryPool, 177
 chThdRelease, 175
EID
 CANRxFrame, 548
 CANTxFrame, 549
el_flags
 EventListener, 574
el_listener
 EventListener, 574
el_mask
 EventListener, 574
el_next
 EventListener, 574
end_cb
 ADCConversionGroup, 502
 SPIConfig, 685
ep0endcb
 USBDriver, 724
ep0n
 USBDriver, 724
ep0next
 USBDriver, 724
ep0state
 USBDriver, 723
EP_STATUS_ACTIVE
 USB Driver, 470
EP_STATUS_DISABLED
 USB Driver, 470
EP_STATUS_STALLED
 USB Driver, 470
ep_mode
 USBEndpointConfig, 726
epc
 USBDriver, 723
error_cb
 ADCConversionGroup, 502
error_event
 CANDriver, 547
errors
 I2CDriver, 593
 SDCDriver, 668
es_next
 EventSource, 576
ETHD1
 MAC Driver, 308
ev_listener
 chibios_rt::EvtListener, 578
ev_source
 chibios_rt::EvtSource, 581
Event Flags, 120
 _EVENTSOURCE_DATA, 131
 ALL_EVENTS, 132
 chEvtAddEvents, 124
 chEvtBroadcast, 133
 chEvtBroadcastFlags, 125
 chEvtBroadcastFlagsl, 126
 chEvtBroadcastl, 133
 chEvtDispatch, 127
 chEvtGetAndClearEvents, 123
 chEvtGetAndClearFlags, 124
 chEvtGetAndClearFlagsl, 124
 chEvtInit, 132
 chEvtIsListeningl, 133
 chEvtRegister, 132
 chEvtRegisterMask, 123
 chEvtSignal, 124
 chEvtSignall, 125
 chEvtUnregister, 123
 chEvtWaitAll, 131
 chEvtWaitAllTimeout, 129
 chEvtWaitAny, 130
 chEvtWaitAnyTimeout, 128
 chEvtWaitOne, 129
 chEvtWaitOneTimeout, 127
 EVENT_MASK, 132
 EventSource, 133
 EVENTSOURCE_DECL, 132
 evhandler_t, 133
event_cb
 USBCConfig, 721
EVENT_MASK
 Event Flags, 132
eventid_t
 Types, 51
EventListener, 572
 el_flags, 574
 el_listener, 574
 el_mask, 574
 el_next, 574
eventmask_t
 Types, 51
EventSource, 574
 es_next, 576
 Event Flags, 133
EVENTSOURCE_DECL

Event Flags, 132
evhandler_t
 Event Flags, 133
EvTimer, 576
evtimer.c, 783
evtimer.h, 783
evtInit
 Periodic Events Timer, 481
EvtSource
 chibios_rt::EvtSource, 578
evtStart
 Periodic Events Timer, 481
evtStop
 Periodic Events Timer, 481
ewmask
 Thread, 696
execute
 testcase, 691
exit
 chibios_rt::BaseThread, 528
exitcode
 Thread, 696
exitS
 chibios_rt::BaseThread, 529
expchannel_t
 EXT Driver, 242
EXT Driver, 233
 expchannel_t, 242
 EXT_ACTIVE, 242
 EXT_STOP, 242
 EXT_UNINIT, 242
 EXT_CH_MODE_AUTOSTART, 241
 EXT_CH_MODE_BOTH_EDGES, 241
 EXT_CH_MODE_DISABLED, 240
 EXT_CH_MODE_EDGES_MASK, 240
 EXT_CH_MODE_FALLING_EDGE, 241
 EXT_CH_MODE_RISING_EDGE, 240
 ext_lld_channel_disable, 240
 ext_lld_channel_enable, 240
 ext_lld_init, 239
 ext_lld_start, 239
 ext_lld_stop, 239
 EXT_MAX_CHANNELS, 242
 extcallback_t, 242
 extChannelDisable, 238
 extChannelDisableI, 241
 extChannelEnable, 238
 extChannelEnableI, 241
 EXTD1, 240
 EXTDriver, 242
 extInit, 236
 extObjectInit, 236
 extSetChannelMode, 241
 extSetChannelModel, 238
 extStart, 237
 extstate_t, 242
 extStop, 237
 PLATFORM_EXT_USE_EXT1, 242
ext.c, 783
ext.h, 784
EXT_ACTIVE
 EXT Driver, 242
EXT_STOP
 EXT Driver, 242
EXT_UNINIT
 EXT Driver, 242
EXT_CH_MODE_AUTOSTART
 EXT Driver, 241
EXT_CH_MODE_BOTH_EDGES
 EXT Driver, 241
EXT_CH_MODE_DISABLED
 EXT Driver, 240
EXT_CH_MODE_EDGES_MASK
 EXT Driver, 240
EXT_CH_MODE_FALLING_EDGE
 EXT Driver, 241
EXT_CH_MODE_RISING_EDGE
 EXT Driver, 240
ext_lld.c, 785
ext_lld.h, 786
ext_lld_channel_disable
 EXT Driver, 240
ext_lld_channel_enable
 EXT Driver, 240
ext_lld_init
 EXT Driver, 239
ext_lld_start
 EXT Driver, 239
ext_lld_stop
 EXT Driver, 239
EXT_MAX_CHANNELS
 EXT Driver, 242
extcallback_t
 EXT Driver, 242
EXTChannelConfig, 581
 cb, 582
 mode, 582
extChannelDisable
 EXT Driver, 238
extChannelDisableI
 EXT Driver, 241
extChannelEnable
 EXT Driver, 238
extChannelEnableI
 EXT Driver, 241
EXTConfig, 582
 channels, 583
extctx, 583
EXTD1
 EXT Driver, 240
EXTDriver, 584
 config, 584
 EXT Driver, 242
 state, 584
External Components, 497
extInit
 EXT Driver, 236
extObjectInit

EXT Driver, 236
extSetChannelMode
 EXT Driver, 241
extSetChannelModel
 EXT Driver, 238
extStart
 EXT Driver, 237
extstate_t
 EXT Driver, 242
extStop
 EXT Driver, 237

FALSE
 Version Numbers and Identification, 39

fetch
 chibios_rt::Mailbox, 615

fetchI
 chibios_rt::Mailbox, 617

fetchS
 chibios_rt::Mailbox, 616

fifo_remove
 Internals, 197

FILE_ERROR
 Abstract File Streams, 182

FILE_OK
 Abstract File Streams, 182

fileoffset_t
 Abstract File Streams, 184

firstprio
 Scheduler, 67

free
 chibios_rt::MemoryPool, 629

freel
 chibios_rt::MemoryPool, 630

frequency
 GPTConfig, 588
 ICUConfig, 594
 PWMConfig, 661

GenericQueue, 585
 I/O Queues, 162
 q_buffer, 587
 q_counter, 587
 q_link, 587
 q_notify, 587
 q_rptr, 587
 q_top, 587
 q_waiting, 587
 q_wptr, 587

get
 chibios_rt::BaseSequentialStreamInterface, 519
 chibios_rt::InQueue, 601

get_descriptor_cb
 USBConfig, 721

getAndClearEvents
 chibios_rt::BaseThread, 531

getAndClearFlags
 chibios_rt::EvtListener, 577

getAndClearFlagsI

 chibios_rt::EvtListener, 577

getCounterI
 chibios_rt::CounterSemaphore, 571

getEmptyI
 chibios_rt::InQueue, 599
 chibios_rt::OutQueue, 652

getFreeCountI
 chibios_rt::Mailbox, 617

getFullI
 chibios_rt::InQueue, 598
 chibios_rt::OutQueue, 651

getI
 chibios_rt::OutQueue, 654

getMessage
 chibios_rt::ThreadReference, 705

getStateI
 chibios_rt::BinarySemaphore, 543

getStatus
 chibios_rt::Core, 564

getTime
 chibios_rt::System, 689

getTimeout
 chibios_rt::InQueue, 601

getUsedCountI
 chibios_rt::Mailbox, 618

GPT Driver, 243
 GPT_CONTINUOUS, 254
 GPT_ONESHOT, 254
 GPT_READY, 254
 GPT_STOP, 254
 GPT_UNINIT, 254
 gpt_lld_change_interval, 253
 gpt_lld_init, 251
 gpt_lld_polled_delay, 252
 gpt_lld_start, 251
 gpt_lld_start_timer, 252
 gpt_lld_stop, 251
 gpt_lld_stop_timer, 252
 gptcallback_t, 254
 gptChangeInterval, 248
 gptChangeIntervall, 252
 gptcnt_t, 254
 GPTD1, 252
 GPTDriver, 254
 gptfreq_t, 254
 gptInit, 245
 gptObjectInit, 246
 gptPolledDelay, 250
 gptStart, 246
 gptStartContinuous, 247
 gptStartContinuousI, 247
 gptStartOneShot, 248
 gptStartOneShotI, 249
 gptstate_t, 254
 gptStop, 246
 gptStopTimer, 249
 gptStopTimerI, 250
 STM32_GPT_USE_TIM1, 253

gpt.c, 786

gpt.h, 787
GPT_CONTINUOUS
 GPT Driver, 254
GPT_ONESHOT
 GPT Driver, 254
GPT_READY
 GPT Driver, 254
GPT_STOP
 GPT Driver, 254
GPT_UNINIT
 GPT Driver, 254
gpt_lld.c, 788
gpt_lld.h, 789
gpt_lld_change_interval
 GPT Driver, 253
gpt_lld_init
 GPT Driver, 251
gpt_lld_polled_delay
 GPT Driver, 252
gpt_lld_start
 GPT Driver, 251
gpt_lld_start_timer
 GPT Driver, 252
gpt_lld_stop
 GPT Driver, 251
gpt_lld_stop_timer
 GPT Driver, 252
gptcallback_t
 GPT Driver, 254
gptChangeInterval
 GPT Driver, 248
gptChangeIntervall
 GPT Driver, 252
gptcnt_t
 GPT Driver, 254
GPTConfig, 588
 callback, 588
 frequency, 588
GPTD1
 GPT Driver, 252
GPTDriver, 589
 config, 589
 GPT Driver, 254
 state, 589
gptfreq_t
 GPT Driver, 254
gptInit
 GPT Driver, 245
gptObjectInit
 GPT Driver, 246
gptPolledDelay
 GPT Driver, 250
gptStart
 GPT Driver, 246
gptStartContinuous
 GPT Driver, 247
gptStartContinuousl
 GPT Driver, 247
gptStartOneShot

 GPT Driver, 248
gptStartOneShotl
 GPT Driver, 249
gptstate_t
 GPT Driver, 254
gptStop
 GPT Driver, 246
gptStopTimer
 GPT Driver, 249
gptStopTimerl
 GPT Driver, 250
grpp
 ADCDriver, 504

h_free
 memory_heap, 624
h_mtx
 memory_heap, 624
h_provider
 memory_heap, 624
HAL, 471
HAL Driver, 254
 HAL_IMPLEMENTANTS_COUNTERS, 262
 hal_lld_get_counter_frequency, 262
 hal_lld_get_counter_value, 262
 hal_lld_init, 259
 halclock_t, 262
 halGetCounterFrequency, 261
 halGetCounterValue, 261
 hallInit, 256
 hallsCounterWithin, 257
 halPolledDelay, 258
 halrtcnt_t, 262
 MS2RTT, 259
 platform_early_init, 259
 RTT2MS, 261
 RTT2S, 260
 RTT2US, 261
 S2RTT, 259
 US2RTT, 260
hal.c, 790
hal.h, 790
HAL_IMPLEMENTANTS_COUNTERS
 HAL Driver, 262
hal_lld.c, 791
hal_lld.h, 791
hal_lld_get_counter_frequency
 HAL Driver, 262
hal_lld_get_counter_value
 HAL Driver, 262
hal_lld_init
 HAL Driver, 259
HAL_USE_ADC
 Configuration, 475
HAL_USE_CAN
 Configuration, 475
HAL_USE_EXT
 Configuration, 475
HAL_USE_GPT

Configuration, 475
HAL_USE_I2C
 Configuration, 476
HAL_USE_ICU
 Configuration, 476
HAL_USE_MAC
 Configuration, 476
HAL_USE_MMCSPI
 Configuration, 476
HAL_USE_PAL
 Configuration, 475
HAL_USE_PWM
 Configuration, 476
HAL_USE_RTC
 Configuration, 476
HAL_USE_SDC
 Configuration, 476
HAL_USE_SERIAL
 Configuration, 476
HAL_USE_SERIAL_USB
 Configuration, 476
HAL_USE_SPI
 Configuration, 476
HAL_USE_TM
 Configuration, 475
HAL_USE_UART
 Configuration, 476
HAL_USE_USB
 Configuration, 476
halclock_t
 HAL Driver, 262
halconf.h, 792
halGetCounterFrequency
 HAL Driver, 261
halGetCounterValue
 HAL Driver, 261
halInit
 HAL Driver, 256
hallsCounterWithin
 HAL Driver, 257
halPolledDelay
 HAL Driver, 258
halrtcnt_t
 HAL Driver, 262
heap
 heap_header, 590
heap_header, 590
 heap, 590
 next, 590
 size, 590
 u, 590
Heaps, 166
 _heap_init, 167
 chHeapAlloc, 168
 chHeapFree, 168
 chHeapInit, 167
 chHeapStatus, 169
HIGHPRIO
 Scheduler, 66
hscfg
 MMCConfig, 634
I/O Queues, 148
 _INPUTQUEUE_DATA, 159
 _OUTPUTQUEUE_DATA, 161
chIQGet, 158
chIQGetEmpty, 157
chIQGetFull, 157
chIQGetTimeout, 152
chIQInit, 150
chIQIsEmpty, 158
chIQIsFull, 158
chIQPutl, 151
chIQReadTimeout, 152
chIQResetl, 150
chOQGetEmpty, 160
chOQGetFull, 159
chOQGetl, 154
chOQInit, 153
chOQIsEmpty, 160
chOQIsFull, 161
chOQPut, 161
chOQPutTimeout, 154
chOQResetl, 153
chOQWriteTimeout, 155
chQGetLink, 157
chQSizeL, 156
chQSpaceL, 156
GenericQueue, 162
InputQueue, 162
INPUTQUEUE_DECL, 159
OutputQueue, 162
OUTPUTQUEUE_DECL, 162
Q_EMPTY, 156
Q_FULL, 156
Q_OK, 156
Q_RESET, 156
Q_TIMEOUT, 156
qnotify_t, 162
I2C Driver, 263
 I2C_ACTIVE_RX, 274
 I2C_ACTIVE_TX, 274
 I2C_READY, 274
 I2C_STOP, 274
 I2C_UNINIT, 274
 i2c_lld_get_errors, 273
 i2c_lld_init, 270
 i2c_lld_master_receive_timeout, 271
 i2c_lld_master_transmit_timeout, 271
 i2c_lld_start, 270
 i2c_lld_stop, 271
 I2C_USE_MUTUAL_EXCLUSION, 273
 i2cAcquireBus, 269
 i2caddr_t, 274
 I2CD1, 272
 I2CD_ACK_FAILURE, 272
 I2CD_ARBITRATION_LOST, 272
 I2CD_BUS_ERROR, 272

I2CD_NO_ERROR, 272
I2CD_OVERRUN, 272
I2CD_PEC_ERROR, 272
I2CD_SMB_ALERT, 273
I2CD_TIMEOUT, 273
I2CDriver, 274
i2cflags_t, 274
i2cGetErrors, 267
i2cInit, 265
i2cMasterReceive, 273
i2cMasterReceiveTimeout, 268
i2cMasterTransmit, 273
i2cMasterTransmitTimeout, 267
i2cObjectInit, 266
i2cReleaseBus, 269
i2cStart, 266
i2cstate_t, 274
i2cStop, 267
PLATFORM_I2C_USE_I2C1, 273
i2c.c, 794
i2c.h, 794
I2C_ACTIVE_RX
 I2C Driver, 274
I2C_ACTIVE_TX
 I2C Driver, 274
I2C_READY
 I2C Driver, 274
I2C_STOP
 I2C Driver, 274
I2C_UNINIT
 I2C Driver, 274
i2c_lld.c, 796
i2c_lld.h, 796
i2c_lld_get_errors
 I2C Driver, 273
i2c_lld_init
 I2C Driver, 270
i2c_lld_master_receive_timeout
 I2C Driver, 271
i2c_lld_master_transmit_timeout
 I2C Driver, 271
i2c_lld_start
 I2C Driver, 270
i2c_lld_stop
 I2C Driver, 271
I2C_USE_MUTUAL_EXCLUSION
 Configuration, 477
 I2C Driver, 273
i2cAcquireBus
 I2C Driver, 269
i2caddr_t
 I2C Driver, 274
I2CConfig, 591
I2CD1
 I2C Driver, 272
I2CD_ACK_FAILURE
 I2C Driver, 272
I2CD_ARBITRATION_LOST
 I2C Driver, 272
I2CD_BUS_ERROR
 I2C Driver, 272
I2CD_NO_ERROR
 I2C Driver, 272
I2CD_OVERRUN
 I2C Driver, 272
I2CD_PEC_ERROR
 I2C Driver, 272
I2CD_SMB_ALERT
 I2C Driver, 273
I2CD_TIMEOUT
 I2C Driver, 273
I2CDriver, 591
 config, 593
 errors, 593
 I2C Driver, 274
 mutex, 593
 state, 593
i2cflags_t
 I2C Driver, 274
i2cGetErrors
 I2C Driver, 267
i2cInit
 I2C Driver, 265
i2cMasterReceive
 I2C Driver, 273
i2cMasterReceiveTimeout
 I2C Driver, 268
i2cMasterTransmit
 I2C Driver, 273
i2cMasterTransmitTimeout
 I2C Driver, 267
i2cObjectInit
 I2C Driver, 266
i2cReleaseBus
 I2C Driver, 269
i2cStart
 I2C Driver, 266
i2cstate_t
 I2C Driver, 274
i2cStop
 I2C Driver, 267
I2S Driver, 274
ICU Driver, 274
 _icu_isr_invoke_overflow_cb, 283
 _icu_isr_invoke_period_cb, 283
 _icu_isr_invoke_width_cb, 283
 ICU_ACTIVE, 285
 ICU_IDLE, 285
 ICU_INPUT_ACTIVE_HIGH, 285
 ICU_INPUT_ACTIVE_LOW, 285
 ICU_READY, 285
 ICU_STOP, 284
 ICU_UNINIT, 284
 ICU_WAITING, 285
 icu_lld_disable, 281
 icu_lld_enable, 280
 icu_lld_get_period, 281
 icu_lld_get_width, 281

icu_lld_init, 280
icu_lld_start, 280
icu_lld_stop, 280
icucallback_t, 284
icucnt_t, 284
ICUD1, 282
icuDisable, 279
icuDisableI, 282
ICUDriver, 284
icuEnable, 279
icuEnableI, 282
icufreq_t, 284
icuGetPeriod, 282
icuGetWidth, 282
icuInit, 277
icumode_t, 285
icuObjectInit, 277
icuStart, 278
icustate_t, 284
icuStop, 278
PLATFORM_ICU_USE_ICU1, 284
icu.c, 797
icu.h, 798
ICU_ACTIVE
 ICU Driver, 285
ICU_IDLE
 ICU Driver, 285
ICU_INPUT_ACTIVE_HIGH
 ICU Driver, 285
ICU_INPUT_ACTIVE_LOW
 ICU Driver, 285
ICU_READY
 ICU Driver, 285
ICU_STOP
 ICU Driver, 284
ICU_UNINIT
 ICU Driver, 284
ICU_WAITING
 ICU Driver, 285
icu_lld.c, 799
icu_lld.h, 799
icu_lld_disable
 ICU Driver, 281
icu_lld_enable
 ICU Driver, 280
icu_lld_get_period
 ICU Driver, 281
icu_lld_get_width
 ICU Driver, 281
icu_lld_init
 ICU Driver, 280
icu_lld_start
 ICU Driver, 280
icu_lld_stop
 ICU Driver, 280
icucallback_t
 ICU Driver, 284
icucnt_t
 ICU Driver, 284
ICUConfig, 593
 frequency, 594
 mode, 594
 overflow_cb, 595
 period_cb, 595
 width_cb, 594
ICUD1
 ICU Driver, 282
icuDisable
 ICU Driver, 279
icuDisableI
 ICU Driver, 282
ICUDriver, 595
 config, 596
 ICU Driver, 284
 state, 596
icuEnable
 ICU Driver, 279
icuEnableI
 ICU Driver, 282
icufreq_t
 ICU Driver, 284
icuGetPeriod
 ICU Driver, 282
icuGetWidth
 ICU Driver, 282
icuInit
 ICU Driver, 277
icumode_t
 ICU Driver, 285
icuObjectInit
 ICU Driver, 277
icuStart
 ICU Driver, 278
icustate_t
 ICU Driver, 284
icuStop
 ICU Driver, 278
IDE
 CANRxFrame, 548
 CANTxFrame, 549
IDLE_LOOP_HOOK
 Configuration, 48
IDLEPRIORITY
 Scheduler, 66
in
 USB Driver, 460
 usb_lld.c, 857
in_cb
 USBEndpointConfig, 726
in_maxsize
 USBEndpointConfig, 726
in_params
 USBDriver, 723
in_state
 USBEndpointConfig, 727
init
 chibios_rt::System, 688
INLINE

Types, 50
InputQueue
 I/O Queues, 162
INPUTQUEUE_DECL
 I/O Queues, 159
InQueue
 chibios_rt::InQueue, 598
InQueueBuffer
 chibios_rt::InQueueBuffer, 604
int_in
 SerialUSBConfig, 677
intctx, 605
Internals, 196
 _THREADSQUEUE_DATA, 199
 dequeue, 198
 fifo_remove, 197
 isempty, 199
 lifo_remove, 197
 list_init, 199
 list_insert, 198
 list_remove, 198
 notempty, 199
 prio_insert, 197
 queue_init, 199
 queue_insert, 197
 _THREADSQUEUE_DECL, 199
io_block.h, 800
io_channel.h, 801
IOBus, 605
 mask, 605
 offset, 605
 portid, 605
IOBUS_DECL
 PAL Driver, 334
iomode_t
 PAL Driver, 345
IOPORT1
 PAL Driver, 340
ioportid_t
 PAL Driver, 345
ioportmask_t
 PAL Driver, 345
isArmed
 chibios_rt::Timer, 715
isempty
 Internals, 199
isEmpty
 chibios_rt::InQueue, 599
 chibios_rt::OutQueue, 652
isFull
 chibios_rt::InQueue, 599
 chibios_rt::OutQueue, 652
isPendingMessage
 chibios_rt::ThreadReference, 705
isTimeWithin
 chibios_rt::System, 690
Kernel, 37
last
 TimeMeasurement, 713
lifo_remove
 Internals, 197
list_init
 Internals, 199
list_insert
 Internals, 198
list_remove
 Internals, 198
loadArray
 chibios_rt::MemoryPool, 627
lock
 chibios_rt::Mutex, 646
 chibios_rt::System, 688
lockFromIsr
 chibios_rt::System, 689
lockS
 chibios_rt::Mutex, 646
LOWPrio
 Scheduler, 66
lscfg
 MMCConfig, 634
m_next
 Mutex, 643
m_owner
 Mutex, 643
m_queue
 Mutex, 643
MAC Driver, 297
 ETHD1, 308
 MAC_ACTIVE, 310
 MAC_STOP, 310
 MAC_UNINIT, 310
 mac_lld_get_next_receive_buffer, 307
 mac_lld_get_next_transmit_buffer, 307
 mac_lld_get_receive_descriptor, 305
 mac_lld_get_transmit_descriptor, 305
 mac_lld_init, 304
 mac_lld_poll_link_status, 306
 mac_lld_read_receive_descriptor, 307
 mac_lld_release_receive_descriptor, 306
 mac_lld_release_transmit_descriptor, 305
 mac_lld_start, 304
 mac_lld_stop, 304
 mac_lld_write_transmit_descriptor, 306
 MAC_SUPPORTS_ZERO_COPY, 310
 MAC_USE_EVENTS, 308
 MAC_USE_ZERO_COPY, 308
 MACDriver, 310
 macGetNextReceiveBuffer, 309
 macGetNextTransmitBuffer, 309
 macGetReceiveEventSource, 308
 macInit, 299
 macObjectInit, 300
 macPollLinkStatus, 303
 macReadReceiveDescriptor, 309
 macReleaseReceiveDescriptor, 303

macReleaseTransmitDescriptor, 302
macStart, 300
macstate_t, 310
macStop, 301
macWaitReceiveDescriptor, 302
macWaitTransmitDescriptor, 301
macWriteTransmitDescriptor, 308
PLATFORM_MAC_USE_MAC1, 310
mac.c, 802
mac.h, 803
MAC_ACTIVE
 MAC Driver, 310
MAC_STOP
 MAC Driver, 310
MAC_UNINIT
 MAC Driver, 310
mac_address
 MACConfig, 606
mac_lld.c, 804
mac_lld.h, 805
mac_lld_get_next_receive_buffer
 MAC Driver, 307
mac_lld_get_next_transmit_buffer
 MAC Driver, 307
mac_lld_get_receive_descriptor
 MAC Driver, 305
mac_lld_get_transmit_descriptor
 MAC Driver, 305
mac_lld_init
 MAC Driver, 304
mac_lld_poll_link_status
 MAC Driver, 306
mac_lld_read_receive_descriptor
 MAC Driver, 307
mac_lld_release_receive_descriptor
 MAC Driver, 306
mac_lld_release_transmit_descriptor
 MAC Driver, 305
mac_lld_start
 MAC Driver, 304
mac_lld_stop
 MAC Driver, 304
mac_lld_write_transmit_descriptor
 MAC Driver, 306
MAC_SUPPORTS_ZERO_COPY
 MAC Driver, 310
MAC_USE_EVENTS
 Configuration, 477
 MAC Driver, 308
MAC_USE_ZERO_COPY
 Configuration, 477
 MAC Driver, 308
MACConfig, 606
 mac_address, 606
MACDriver, 606
 config, 607
 MAC Driver, 310
 rdevent, 608
 rdsem, 608
 state, 607
 tdsem, 608
macGetNextReceiveBuffer
 MAC Driver, 309
macGetNextTransmitBuffer
 MAC Driver, 309
macGetReceiveEventSource
 MAC Driver, 308
macInit
 MAC Driver, 299
macObjectInit
 MAC Driver, 300
macPollLinkStatus
 MAC Driver, 303
macReadReceiveDescriptor
 MAC Driver, 309
MACReceiveDescriptor, 608
 offset, 608
 size, 608
macReleaseReceiveDescriptor
 MAC Driver, 303
macReleaseTransmitDescriptor
 MAC Driver, 302
macStart
 MAC Driver, 300
macstate_t
 MAC Driver, 310
macStop
 MAC Driver, 301
MACTransmitDescriptor, 608
 offset, 609
 size, 609
macWaitReceiveDescriptor
 MAC Driver, 302
macWaitTransmitDescriptor
 MAC Driver, 301
macWriteTransmitDescriptor
 MAC Driver, 308
Mailbox, 618
 chibios_rt::Mailbox, 611
 mb_buffer, 619
 mb_emptysem, 620
 mb_fullsem, 620
 mb_rptr, 620
 mb_top, 620
 mb_wptr, 620
MAILBOX_DECL
 Mailboxes, 148
MailboxBuffer
 chibios_rt::MailboxBuffer, 622
Mailboxes, 137
 _MAILBOX_DATA, 147
 chMBFetch, 144
 chMBFetchl, 145
 chMBFetchS, 144
 chMBGetFreeCountl, 146
 chMBGetUsedCountl, 147
 chMBInit, 138
 chMBPeekl, 147

chMBPost, 139
chMBPostAhead, 141
chMBPostAheadl, 143
chMBPostAheadS, 142
chMBPostl, 141
chMBPostS, 140
chMBReset, 139
chMBSizel, 146
MAILBOX_DECL, 148
main
 chibios_rt::BaseThread, 527
mask
 IOBus, 605
mb
 chibios_rt::Mailbox, 618
mb_buffer
 Mailbox, 619
mb_emptysem
 Mailbox, 620
mb_fullsem
 Mailbox, 620
mb_rptr
 Mailbox, 620
mb_top
 Mailbox, 620
mb_wptr
 Mailbox, 620
MEM_ALIGN_MASK
 Core Memory Manager, 166
MEM_ALIGN_NEXT
 Core Memory Manager, 166
MEM_ALIGN_PREV
 Core Memory Manager, 166
MEM_ALIGN_SIZE
 Core Memory Manager, 166
MEM_IS_ALIGNED
 Core Memory Manager, 166
memgetfunc_t
 Core Memory Manager, 166
Memory Management, 163
Memory Pools, 169
 _MEMORYPOOL_DATA, 173
 chPoolAdd, 174
 chPoolAddl, 174
 chPoolAlloc, 171
 chPoolAllocl, 171
 chPoolFree, 173
 chPoolFreel, 172
 chPoolInit, 170
 chPoolLoadArray, 170
 MEMORYPOOL_DECL, 173
Memory Streams, 479
 _memory_stream_data, 480
 msObjectInit, 480
memory_heap, 622
 h_free, 624
 h_mtx, 624
 h_provider, 624
MemoryPool, 624
 chibios_rt::MemoryPool, 626, 627
 mp_next, 625
 mp_object_size, 625
 mp_provider, 625
MEMORYPOOL_DECL
 Memory Pools, 173
MemoryStream, 630
 vmt, 632
memstreams.c, 806
memstreams.h, 806
MemStreamVMT, 632
MMC over SPI Driver, 310
 mmc_driver_methods, 321
 MMC_NICE_WAITING, 321
 mmcConnect, 313
 mmcDisconnect, 314
 mmcErase, 320
 mmcGetInfo, 320
 mmclInit, 312
 mmclsCardInserted, 321
 mmclsWriteProtected, 322
 mmcObjectInit, 312
 mmcSequentialRead, 316
 mmcSequentialWrite, 318
 mmcStart, 312
 mmcStartSequentialRead, 315
 mmcStartSequentialWrite, 317
 mmcStop, 312
 mmcStopSequentialRead, 316
 mmcStopSequentialWrite, 319
 mmcSync, 320
MMC/SD Block Device, 322
 _mmcsd_block_device_data, 326
 _mmcsd_block_device_methods, 326
 MMCSD_BLOCK_SIZE, 325
 MMCSD_CMD8_PATTERN, 325
 MMCSD_CSD_20_CRC_SLICE, 325
 MMCSD_R1_ERROR, 326
 MMCSD_R1_ERROR_MASK, 325
 MMCSD_R1_IS_CARD_LOCKED, 326
 MMCSD_R1_STS, 326
 mmcsdGetCapacity, 325
 mmcsdGetCardCapacity, 326
MMC_NICE_WAITING
 Configuration, 477
 MMC over SPI Driver, 321
mmc_spi.c, 807
mmc_spi.h, 808
MMConfig, 632
 hscfg, 634
 lscfg, 634
 spip, 634
mmcConnect
 MMC over SPI Driver, 313
mmcDisconnect
 MMC over SPI Driver, 314
MMCDriver, 634
 config, 636
 vmt, 636

MMCDriverVMT, 637
mmcErase
 MMC over SPI Driver, 320
mmcGetInfo
 MMC over SPI Driver, 320
mmcInit
 MMC over SPI Driver, 312
mmclsCardInserted
 MMC over SPI Driver, 321
mmclsWriteProtected
 MMC over SPI Driver, 322
mmcObjectInit
 MMC over SPI Driver, 312
mmcsd.c, 809
mmcsd.h, 809
MMCSD_BLOCK_SIZE
 MMC/SD Block Device, 325
MMCSD_CMD8_PATTERN
 MMC/SD Block Device, 325
MMCSD_CSD_20_CRC_SLICE
 MMC/SD Block Device, 325
MMCSD_R1_ERROR
 MMC/SD Block Device, 326
 SDC Driver, 380
MMCSD_R1_ERROR_MASK
 MMC/SD Block Device, 325
MMCSD_R1_IS_CARD_LOCKED
 MMC/SD Block Device, 326
 SDC Driver, 381
MMCSD_R1_STS
 MMC/SD Block Device, 326
 SDC Driver, 381
MMCSDBlockDevice, 638
 vmt, 640
MMCSDBlockDeviceVMT, 640
mmcsdGetCapacity
 MMC/SD Block Device, 325
mmcsdGetCardCapacity
 MMC/SD Block Device, 326
mmcSequentialRead
 MMC over SPI Driver, 316
mmcSequentialWrite
 MMC over SPI Driver, 318
mmcStart
 MMC over SPI Driver, 312
mmcStartSequentialRead
 MMC over SPI Driver, 315
mmcStartSequentialWrite
 MMC over SPI Driver, 317
mmcStop
 MMC over SPI Driver, 312
mmcStopSequentialRead
 MMC over SPI Driver, 316
mmcStopSequentialWrite
 MMC over SPI Driver, 319
mmcSync
 MMC over SPI Driver, 320
mode
 EXTChannelConfig, 582
 ICUConfig, 594
 PWMChannelConfig, 660
mp_next
 MemoryPool, 625
mp_object_size
 MemoryPool, 625
mp_provider
 MemoryPool, 625
MS2RTT
 HAL Driver, 259
MS2ST
 Time and Virtual Timers, 86
msg_t
 Types, 50
msObjectInit
 Memory Streams, 480
Mutex, 641
 chibios_rt::Mutex, 644
 m_next, 643
 m_owner, 643
 m_queue, 643
 Mutexes, 113
mutex
 ADCDriver, 504
 chibios_rt::Mutex, 647
 I2CDriver, 593
 SPIDriver, 687
MUTEX_DECL
 Mutexes, 112
Mutexes, 105
 _MUTEX_DATA, 112
 chMtxInit, 107
 chMtxLock, 107
 chMtxLockS, 108
 chMtxQueueNotEmptyS, 112
 chMtxTryLock, 108
 chMtxTryLockS, 109
 chMtxUnlock, 110
 chMtxUnlockAll, 111
 chMtxUnlockS, 111
 Mutex, 113
 MUTEX_DECL, 112
name
 testcase, 690
next
 heap_header, 590
NOPRIO
 Scheduler, 66
NORMALPRIO
 Scheduler, 66
notempty
 Internals, 199
num_channels
 ADCCConversionGroup, 502
ObjectsPool
 chibios_rt::ObjectsPool, 649
offset

IOBus, 605
MACReceiveDescriptor, 608
MACTransmitDescriptor, 609
out
 USB Driver, 460
 usb_lld.c, 857
out_cb
 USBEndpointConfig, 726
out_maxsize
 USBEndpointConfig, 726
out_params
 USBDriver, 723
out_state
 USBEndpointConfig, 727
OutputQueue
 I/O Queues, 162
OUTPUTQUEUE_DECL
 I/O Queues, 162
OutQueue
 chibios_rt::OutQueue, 651
OutQueueBuffer
 chibios_rt::OutQueueBuffer, 657
overflow_cb
 ICUConfig, 595

p_ctx
 Thread, 695
p_epending
 Thread, 697
p_flags
 Thread, 695
p_mpool
 Thread, 697
p_msg
 Thread, 697
p_msgqueue
 Thread, 696
p_mtxlist
 Thread, 697
p_name
 Thread, 695
p_newer
 Thread, 695
p_next
 Thread, 695
 ThreadsList, 710
 ThreadsQueue, 712
p_older
 Thread, 695
p_preempt
 Thread, 695
p_prev
 Thread, 695
 ThreadsQueue, 713
p_prio
 Thread, 695
p_realprio
 Thread, 697
p_refs
 Thread, 695
 Thread, 695
p_state
 Thread, 695
p_stklimit
 Thread, 695
p_time
 Thread, 696
p_waiting
 Thread, 696
PACK_STRUCT_BEGIN
 Types, 50
PACK_STRUCT_END
 Types, 50
PACK_STRUCT_STRUCT
 Types, 50
PAL Driver, 327
 _IOBUS_DATA, 334
 _pal_lld_init, 332
 _pal_lld_setgroupmode, 332
 IOBUS_DECL, 334
 iomode_t, 345
 IOPORT1, 340
 ioportid_t, 345
 ioportmask_t, 345
 PAL_GROUP_MASK, 333
 PAL_HIGH, 333
 PAL_IOPORTS_WIDTH, 340
 pal_lld_clearpad, 344
 pal_lld_clearport, 341
 pal_lld_init, 340
 pal_lld_readgroup, 342
 pal_lld_readdlatch, 341
 pal_lld_readpad, 343
 pal_lld_readport, 340
 pal_lld_setgroupmode, 343
 pal_lld_setpad, 344
 pal_lld_setpadmode, 345
 pal_lld_setport, 341
 pal_lld_togglepad, 345
 pal_lld_toggleport, 342
 pal_lld_writegroup, 342
 pal_lld_writepad, 344
 pal_lld_writeport, 341
 PAL_LOW, 333
 PAL_MODE_INPUT, 332
 PAL_MODE_INPUT_ANALOG, 333
 PAL_MODE_INPUT_PULLDOWN, 333
 PAL_MODE_INPUT_PULLUP, 332
 PAL_MODE_OUTPUT_OPENDRAIN, 333
 PAL_MODE_OUTPUT_PUSH_PULL, 333
 PAL_MODE_RESET, 332
 PAL_MODE_UNCONNECTED, 332
 PAL_PORT_BIT, 333
 PAL_WHOLE_PORT, 340
 palClearPad, 339
 palClearPort, 336
 pallInit, 334
 palReadBus, 331
 palReadGroup, 336

palReadLatch, 335
palReadPad, 337
palReadPort, 334
palSetBusMode, 331
palSetGroupMode, 337
palSetPad, 338
palSetPadMode, 339
palSetPort, 335
palTogglePad, 339
palTogglePort, 336
palWriteBus, 331
palWriteGroup, 337
palWritePad, 338
palWritePort, 335
pal.c, 812
pal.h, 812
PAL_GROUP_MASK
 PAL Driver, 333
PAL_HIGH
 PAL Driver, 333
PAL_IOPORTS_WIDTH
 PAL Driver, 340
pal_lld.c, 814
pal_lld.h, 814
pal_lld_clearpad
 PAL Driver, 344
pal_lld_clearport
 PAL Driver, 341
pal_lld_init
 PAL Driver, 340
pal_lld_readgroup
 PAL Driver, 342
pal_lld_readdlatch
 PAL Driver, 341
pal_lld_readpad
 PAL Driver, 343
pal_lld_readport
 PAL Driver, 340
pal_lld_setgroupmode
 PAL Driver, 343
pal_lld_setpad
 PAL Driver, 344
pal_lld_setpadmode
 PAL Driver, 345
pal_lld_setport
 PAL Driver, 341
pal_lld_togglepad
 PAL Driver, 345
pal_lld_toggleport
 PAL Driver, 342
pal_lld_writegroup
 PAL Driver, 342
pal_lld_writepad
 PAL Driver, 344
pal_lld_writeport
 PAL Driver, 341
PAL_LOW
 PAL Driver, 333
PAL_MODE_INPUT

PAL Driver, 332
PAL_MODE_INPUT_ANALOG
 PAL Driver, 333
PAL_MODE_INPUT_PULLDOWN
 PAL Driver, 333
PAL_MODE_INPUT_PULLUP
 PAL Driver, 332
PAL_MODE_OUTPUT_OPENDRAIN
 PAL Driver, 333
PAL_MODE_OUTPUT_PUSHPULL
 PAL Driver, 333
PAL_MODE_RESET
 PAL Driver, 332
PAL_MODE_UNCONNECTED
 PAL Driver, 332
PAL_PORT_BIT
 PAL Driver, 333
PAL_WHOLE_PORT
 PAL Driver, 340
palClearPad
 PAL Driver, 339
palClearPort
 PAL Driver, 336
PALConfig, 658
pallInit
 PAL Driver, 334
palReadBus
 PAL Driver, 331
palReadGroup
 PAL Driver, 336
palReadLatch
 PAL Driver, 335
palReadPad
 PAL Driver, 337
palReadPort
 PAL Driver, 334
palSetBusMode
 PAL Driver, 331
palSetGroupMode
 PAL Driver, 337
palSetPad
 PAL Driver, 338
palSetPadMode
 PAL Driver, 339
palSetPort
 PAL Driver, 335
palTogglePad
 PAL Driver, 339
palTogglePort
 PAL Driver, 336
palWriteBus
 PAL Driver, 331
palWriteGroup
 PAL Driver, 337
palWritePad
 PAL Driver, 338
palWritePort
 PAL Driver, 335
patternbmk

testbmk.c, 838
testbmk.h, 839
patterndyn
 testdyn.c, 839
 testdyn.h, 840
patternevt
 testevt.c, 840
 testevt.h, 840
patternheap
 testheap.c, 841
 testheap.h, 841
patternmbox
 testmbox.c, 842
 testmbox.h, 842
patternmsg
 testmsg.c, 843
 testmsg.h, 843
patternmtx
 testmtx.c, 843
 testmtx.h, 844
patternqueues
 testqueues.c, 845
 testqueues.h, 845
patternsem
 testsem.c, 846
 testsem.h, 846
patternthd
 testthd.c, 847
 testthd.h, 847
period
 PWMConfig, 661
 PWMDriver, 663
period_cb
 ICUConfig, 595
Periodic Events Timer, 480
 evtInit, 481
 evtStart, 481
 evtStop, 481
ph_next
 pool_header, 658
PLATFORM_ADC_USE_ADC1
 ADC Driver, 220
PLATFORM_CAN_USE_CAN1
 CAN Driver, 233
platform_early_init
 HAL Driver, 259
PLATFORM_EXT_USE_EXT1
 EXT Driver, 242
PLATFORM_I2C_USE_I2C1
 I2C Driver, 273
PLATFORM_ICU_USE_ICU1
 ICU Driver, 284
PLATFORM_MAC_USE_MAC1
 MAC Driver, 310
PLATFORM_PWM_USE_PWM1
 PWM Driver, 358
PLATFORM_SDC_USE_SDC1
 SDC Driver, 380
PLATFORM_SERIAL_USE_SD1
 Serial Driver, 393
PLATFORM_SPI_USE_SPI1
 SPI Driver, 420
PLATFORM_UART_USE_UART1
 UART Driver, 436
PLATFORM_USB_USE_USB1
 USB Driver, 468
pool
 chibios_rt::MemoryPool, 630
pool_header, 658
 ph_next, 658
Port, 200
 CH_ARCHITECTURE_NAME, 203
 CH_ARCHITECTURE_VARIANT_NAME, 203
 CH_ARCHITECTURE_XXX, 203
 CH_COMPILER_NAME, 204
 CH_PORT_INFO, 204
 port_disable, 202
 port_enable, 202
 PORT_FAST_IRQ_HANDLER, 205
 port_halt, 202
 PORT_IDLE_THREAD_STACK_SIZE, 203
 port_init, 201
 PORT_INT_REQUIRED_STACK, 203
 PORT_IRQ_EPILOGUE, 204
 PORT_IRQ_HANDLER, 204
 PORT_IRQ_PROLOGUE, 204
 port_lock, 201
 port_lock_from_isr, 201
 port_suspend, 202
 port_switch, 203
 port_unlock, 201
 port_unlock_from_isr, 202
 port_wait_for_interrupt, 202
 SETUP_CONTEXT, 204
 STACK_ALIGN, 204
 stkalign_t, 205
 THD_WA_SIZE, 204
 WORKING_AREA, 204
port_disable
 Port, 202
port_enable
 Port, 202
 PORT_FAST_IRQ_HANDLER
 Port, 205
port_halt
 Port, 202
 PORT_IDLE_THREAD_STACK_SIZE
 Port, 203
port_init
 Port, 201
 PORT_INT_REQUIRED_STACK
 Port, 203
 PORT_IRQ_EPILOGUE
 Port, 204
 PORT_IRQ_HANDLER
 Port, 204
 PORT_IRQ_PROLOGUE
 Port, 204

port_lock
 Port, 201
port_lock_from_isr
 Port, 201
port_suspend
 Port, 202
port_switch
 Port, 203
port_unlock
 Port, 201
port_unlock_from_isr
 Port, 202
port_wait_for_interrupt
 Port, 202
portid
 IOBus, 605
post
 chibios_rt::Mailbox, 611
postAhead
 chibios_rt::Mailbox, 613
postAheadI
 chibios_rt::Mailbox, 615
postAheadS
 chibios_rt::Mailbox, 614
postI
 chibios_rt::Mailbox, 613
postS
 chibios_rt::Mailbox, 612
prio_insert
 Internals, 197
put
 chibios_rt::BaseSequentialStreamInterface, 519
 chibios_rt::OutQueue, 653
putl
 chibios_rt::InQueue, 600
putTimeout
 chibios_rt::OutQueue, 653
PWM Driver, 346
 PLATFORM_PWM_USE_PWM1, 358
 PWM_READY, 359
 PWM_STOP, 359
 PWM_UNINIT, 359
 PWM_CHANNELS, 357
 PWM_DEGREES_TO_WIDTH, 355
 PWM_FRACTION_TO_WIDTH, 354
 pwm_lld_change_period, 353
 pwm_lld_disable_channel, 354
 pwm_lld_enable_channel, 353
 pwm_lld_init, 352
 pwm_lld_is_channel_enabled, 358
 pwm_lld_start, 352
 pwm_lld_stop, 352
 PWM_OUTPUT_ACTIVE_HIGH, 354
 PWM_OUTPUT_ACTIVE_LOW, 354
 PWM_OUTPUT_DISABLED, 354
 PWM_OUTPUT_MASK, 354
 PWM_PERCENTAGE_TO_WIDTH, 355
 pwmcallback_t, 358
 pwmChangePeriod, 350
 pwmChangePeriodI, 356
 pwmchannel_t, 358
 pwmcnt_t, 358
 PWMD1, 354
 pwmDisableChannel, 351
 pwmDisableChannell, 357
 PWMDriver, 358
 pwmEnableChannel, 351
 pwmEnableChannell, 356
 pwmInit, 349
 pwmlsChannelEnabledI, 357
 pwmmode_t, 358
 pwmObjectInit, 349
 pwmStart, 349
 pwmstate_t, 359
 pwmStop, 350
pwm.c, 815
pwm.h, 816
PWM_READY
 PWM Driver, 359
PWM_STOP
 PWM Driver, 359
PWM_UNINIT
 PWM Driver, 359
PWM_CHANNELS
 PWM Driver, 357
PWM_DEGREES_TO_WIDTH
 PWM Driver, 355
PWM_FRACTION_TO_WIDTH
 PWM Driver, 354
pwm_lld.c, 817
pwm_lld.h, 818
pwm_lld_change_period
 PWM Driver, 353
pwm_lld_disable_channel
 PWM Driver, 354
pwm_lld_enable_channel
 PWM Driver, 353
pwm_lld_init
 PWM Driver, 352
pwm_lld_is_channel_enabled
 PWM Driver, 358
pwm_lld_start
 PWM Driver, 352
pwm_lld_stop
 PWM Driver, 352
 PWM_OUTPUT_ACTIVE_HIGH
 PWM Driver, 354
 PWM_OUTPUT_ACTIVE_LOW
 PWM Driver, 354
 PWM_OUTPUT_DISABLED
 PWM Driver, 354
 PWM_OUTPUT_MASK
 PWM Driver, 354
 PWM_PERCENTAGE_TO_WIDTH
 PWM Driver, 355
 pwmcallback_t
 PWM Driver, 358
 pwmChangePeriod

PWM Driver, 350
pwmChangePeriodI
 PWM Driver, 356
pwmchannel_t
 PWM Driver, 358
PWMChannelConfig, 659
 callback, 660
 mode, 660
pwmcnt_t
 PWM Driver, 358
PWMConfig, 660
 callback, 662
 channels, 662
 frequency, 661
 period, 661
PWMD1
 PWM Driver, 354
pwmDisableChannel
 PWM Driver, 351
pwmDisableChannell
 PWM Driver, 357
PWMDriver, 662
 config, 663
 period, 663
 PWM Driver, 358
 state, 663
pwmEnableChannel
 PWM Driver, 351
pwmEnableChannell
 PWM Driver, 356
pwmInit
 PWM Driver, 349
pwmlsChannelEnabledI
 PWM Driver, 357
pwmmode_t
 PWM Driver, 358
pwmObjectInit
 PWM Driver, 349
pwmStart
 PWM Driver, 349
pwmstate_t
 PWM Driver, 359
pwmStop
 PWM Driver, 350

q_buffer
 GenericQueue, 587
q_counter
 GenericQueue, 587
Q_EMPTY
 I/O Queues, 156
Q_FULL
 I/O Queues, 156
q_link
 GenericQueue, 587
q_notify
 GenericQueue, 587
Q_OK
 I/O Queues, 156

q_rptr
 GenericQueue, 587
Q_RESET
 I/O Queues, 156
Q_TIMEOUT
 I/O Queues, 156
q_top
 GenericQueue, 587
q_waiting
 GenericQueue, 587
q_wptr
 GenericQueue, 587
qnotify_t
 I/O Queues, 162
queue_init
 Internals, 199
queue_insert
 Internals, 197

r_ctx
 ReadyList, 666
r_current
 ReadyList, 666
r_newer
 ReadyList, 666
r_older
 ReadyList, 666
r_prio
 ReadyList, 666
r_queue
 ReadyList, 666
rca
 SDCDriver, 668
rdevent
 MACDriver, 608
rdsem
 MACDriver, 608
RDY_OK
 Scheduler, 66
RDY_RESET
 Scheduler, 66
RDY_TIMEOUT
 Scheduler, 66
rdymsg
 Thread, 696
read
 chibios_rt::BaseSequentialStreamInterface, 518
readTimeout
 chibios_rt::InQueue, 602
ReadyList, 664
 r_ctx, 666
 r_current, 666
 r_newer, 666
 r_older, 666
 r_prio, 666
 r_queue, 666
receiving
 USBDriver, 723
REG_INSERT

Registry, 187
REG_REMOVE
 Registry, 186
registerMask
 chibios_rt::EvtSource, 579
registerOne
 chibios_rt::EvtSource, 579
Registry, 184
 chRegFirstThread, 185
 chRegGetThreadName, 186
 chRegNextThread, 185
 chRegSetThreadName, 186
 REG_INSERT, 187
 REG_REMOVE, 186
releaseMessage
 chibios_rt::ThreadReference, 705
requests_hook_cb
 USBConfig, 721
requestTerminate
 chibios_rt::ThreadReference, 703
reset
 chibios_rt::BinarySemaphore, 542
 chibios_rt::CounterSemaphore, 566
 chibios_rt::Mailbox, 611
resetl
 chibios_rt::BinarySemaphore, 543
 chibios_rt::CounterSemaphore, 567
 chibios_rt::InQueue, 599
 chibios_rt::OutQueue, 652
 chibios_rt::Timer, 715
resume
 chibios_rt::ThreadReference, 702
resumel
 chibios_rt::ThreadReference, 702
rlist
 Scheduler, 66
ROMCONST
 Types, 50
RTC Driver, 359
 RTCDriver, 363
 rtcGetAlarm, 361
 rtcGetAlarml, 362
 rtcGetTime, 360
 rtcGetTimeFat, 361
 rtcGetTimel, 362
 rtcInit, 360
 rtcSetAlarm, 361
 rtcSetAlarml, 362
 rtcSetCallback, 361
 rtcSetCallbackl, 363
 rtcSetTime, 360
 rtcSetTimel, 362
 RTCTime, 363
RTC time conversion utilities, 486
 rtcGetTimeFat, 489
 rtcGetTimeFatFromCounter, 489
 rtcGetTimeTm, 486
 rtcGetTimeUnixSec, 487
 rtcGetTimeUnixUsec, 488
 rtcSetTimeTm, 487
 rtcSetTimeUnixSec, 488
 rtcSetTimeUnixUsec, 488
 rtcSetTimeTm, 487
 rtcSetTimeUnixSec, 488
 rtc.c, 819
 rtc.h, 819
 RTCDriver
 RTC Driver, 363
 rtcGetAlarm
 RTC Driver, 361
 rtcGetAlarml
 RTC Driver, 362
 rtcGetTime
 RTC Driver, 360
 rtcGetTimeFat
 RTC Driver, 361
 RTC time conversion utilities, 489
 rtcGetTimeFatFromCounter
 RTC time conversion utilities, 489
 rtcGetTimel
 RTC Driver, 362
 rtcGetTimeTm
 RTC time conversion utilities, 486
 rtcGetTimeUnixSec
 RTC time conversion utilities, 487
 rtcGetTimeUnixUsec
 RTC time conversion utilities, 488
 rtcInit
 RTC Driver, 360
 rtcSetAlarm
 RTC Driver, 361
 rtcSetAlarml
 RTC Driver, 362
 rtcSetCallback
 RTC Driver, 361
 rtcSetCallbackl
 RTC Driver, 363
 rtcSetTime
 RTC Driver, 360
 rtcSetTimel
 RTC Driver, 362
 rtcSetTimeTm
 RTC time conversion utilities, 487
 rtcSetTimeUnixSec
 RTC time conversion utilities, 488
RTCTime
 RTC Driver, 363
RTR
 CANRxFrame, 548
 CANTxFrame, 548
RTT2MS
 HAL Driver, 261
RTT2S
 HAL Driver, 260
RTT2US
 HAL Driver, 261
rxbuf
 USBOutEndpointState, 731
rxchar_cb
 UARTConfig, 718
rcnt

USBOutEndpointState, 731
rxend_cb
 UARTConfig, 717
rxerr_cb
 UARTConfig, 718
rfull_event
 CANDriver, 546
rxqueue
 USBOutEndpointState, 731
rxqueued
 USBOutEndpointState, 731
rxsem
 CANDriver, 546
rsize
 USBOutEndpointState, 731
rxstate
 UARTDriver, 719

S2RTT
 HAL Driver, 259
S2ST
 Time and Virtual Timers, 86
s_cnt
 Semaphore, 672
s_queue
 Semaphore, 672
samples
 ADCDriver, 504
sc_channel
 ShellConfig, 682
sc_commands
 ShellConfig, 682
sc_function
 ShellCommand, 681
sc_name
 ShellCommand, 681
sc_speed
 SerialConfig, 672
Scheduler, 59
 _scheduler_init, 61
ABSPRIO, 67
chSchCanYieldS, 68
chSchDoReschedule, 65
chSchDoRescheduleAhead, 65
chSchDoRescheduleBehind, 64
chSchDoYieldS, 68
chSchGoSleepS, 61
chSchGoSleepTimeoutS, 62
chSchIsPreemptionRequired, 64
chSchIsRescRequired, 67
chSchPreemption, 68
chSchReadyI, 61
chSchRescheduleS, 63
chSchWakeups, 63
currp, 67
firstprio, 67
HIGHPRIO, 66
IDLEPRIO, 66
LOWPRIO, 66
 NOPRIO, 66
 NORMALPRIO, 66
 RDY_OK, 66
 RDY_RESET, 66
 RDY_TIMEOUT, 66
 rlist, 66
 setcurrp, 67
 TIME_IMMEDIATE, 67
 TIME_INFINITE, 67
SD1
 Serial Driver, 389
SD_READY
 Serial Driver, 394
SD_STOP
 Serial Driver, 394
SD_UNINIT
 Serial Driver, 394
SD_BREAK_DETECTED
 Serial Driver, 389
SD_FRAMING_ERROR
 Serial Driver, 389
sd_lld_init
 Serial Driver, 388
sd_lld_start
 Serial Driver, 388
sd_lld_stop
 Serial Driver, 388
SD_NOISE_ERROR
 Serial Driver, 389
SD_OVERRUN_ERROR
 Serial Driver, 389
SD_PARITY_ERROR
 Serial Driver, 389
sdAsynchronousRead
 Serial Driver, 393
sdAsynchronousWrite
 Serial Driver, 392
SDC Driver, 363
 _sdc_driver_methods, 380
 _sdc_wait_for_transfer_state, 373
 MMCSD_R1_ERROR, 380
 MMCSD_R1_IS_CARD_LOCKED, 381
 MMCSD_R1_STS, 381
 PLATFORM_SDC_USE_SDC1, 380
 SDC_CMD_CRC_ERROR, 378
 SDC_COMMAND_TIMEOUT, 379
 SDC_DATA_CRC_ERROR, 378
 SDC_DATA_TIMEOUT, 378
 SDC_INIT_RETRY, 379
 sdc_lld_init, 373
 sdc_lld_read, 377
 sdc_lld_send_cmd_long_crc, 376
 sdc_lld_send_cmd_none, 375
 sdc_lld_send_cmd_short, 375
 sdc_lld_send_cmd_short_crc, 376
 sdc_lld_set_bus_mode, 375
 sdc_lld_set_data_clk, 374
 sdc_lld_start, 374
 sdc_lld_start_clk, 374

sdc_lld_stop, 374
sdc_lld_stop_clk, 375
sdc_lld_sync, 377
sdc_lld_write, 377
SDC_MMIC_SUPPORT, 379
SDC_MODE_CARDTYPE_MASK, 378
SDC_MODE_CARDTYPE_MMC, 378
SDC_MODE_CARDTYPE_SDV11, 378
SDC_MODE_CARDTYPE_SDV20, 378
SDC_MODE_HIGH_CAPACITY, 378
SDC_NICE_WAITING, 379
SDC_NO_ERROR, 378
SDC_OVERFLOW_ERROR, 379
SDC_RX_OVERRUN, 379
SDC_STARTBIT_ERROR, 379
SDC_TX_UNDERRUN, 379
sdcbusmode_t, 381
sdcConnect, 368
SDCD1, 378
sdcDisconnect, 369
SDCDriver, 381
sdcErase, 372
sdcfags_t, 381
sdcGetAndClearErrors, 371
sdcGetInfo, 372
sdcInit, 367
sdclIsCardInserted, 379
sdclIsWriteProtected, 380
sdemode_t, 381
sdcObjectInit, 367
sdcRead, 370
sdcStart, 367
sdcStop, 368
sdcSync, 371
sdcWrite, 370
sdc.c, 820
sdc.h, 821
SDC_CMD_CRC_ERROR
 SDC Driver, 378
SDC_COMMAND_TIMEOUT
 SDC Driver, 379
SDC_DATA_CRC_ERROR
 SDC Driver, 378
SDC_DATA_TIMEOUT
 SDC Driver, 378
SDC_INIT_RETRY
 Configuration, 477
 SDC Driver, 379
sdc_lld.c, 822
sdc_lld.h, 823
sdc_lld_init
 SDC Driver, 373
sdc_lld_read
 SDC Driver, 377
sdc_lld_send_cmd_long_crc
 SDC Driver, 376
sdc_lld_send_cmd_none
 SDC Driver, 375
sdc_lld_send_cmd_short
 SDC Driver, 375
sdc_lld_send_cmd_short_crc
 SDC Driver, 376
sdc_lld_set_bus_mode
 SDC Driver, 375
sdc_lld_set_data_clk
 SDC Driver, 374
sdc_lld_start
 SDC Driver, 374
sdc_lld_start_clk
 SDC Driver, 374
sdc_lld_stop
 SDC Driver, 374
sdc_lld_stop_clk
 SDC Driver, 375
sdc_lld_sync
 SDC Driver, 377
sdc_lld_write
 SDC Driver, 377
SDC_MMIC_SUPPORT
 Configuration, 477
 SDC Driver, 379
SDC_MODE_CARDTYPE_MASK
 SDC Driver, 378
SDC_MODE_CARDTYPE_MMC
 SDC Driver, 378
SDC_MODE_CARDTYPE_SDV11
 SDC Driver, 378
SDC_MODE_CARDTYPE_SDV20
 SDC Driver, 378
SDC_MODE_HIGH_CAPACITY
 SDC Driver, 378
SDC_NICE_WAITING
 Configuration, 478
 SDC Driver, 379
SDC_NO_ERROR
 SDC Driver, 378
SDC_OVERFLOW_ERROR
 SDC Driver, 379
SDC_RX_OVERRUN
 SDC Driver, 379
SDC_STARTBIT_ERROR
 SDC Driver, 379
SDC_TX_UNDERRUN
 SDC Driver, 379
sdcbusmode_t
 SDC Driver, 381
SDCCConfig, 666
sdcConnect
 SDC Driver, 368
SDCD1
 SDC Driver, 378
sdcDisconnect
 SDC Driver, 369
SDCDriver, 667
 cardmode, 668
 config, 668
 errors, 668
 rca, 668

SDC Driver, 381
vmt, 668
SDCDriverVMT, 668
sdcErase
 SDC Driver, 372
sdcflags_t
 SDC Driver, 381
sdcGetAndClearErrors
 SDC Driver, 371
sdcGetInfo
 SDC Driver, 372
sdcInit
 SDC Driver, 367
sdclsCardInserted
 SDC Driver, 379
sdclsWriteProtected
 SDC Driver, 380
sdemode_t
 SDC Driver, 381
sdcObjectInit
 SDC Driver, 367
sdcRead
 SDC Driver, 370
sdcStart
 SDC Driver, 367
sdcStop
 SDC Driver, 368
sdcSync
 SDC Driver, 371
sdcWrite
 SDC Driver, 370
sdGet
 Serial Driver, 391
sdGetTimeout
 Serial Driver, 391
sdGetWouldBlock
 Serial Driver, 390
sdIncomingData
 Serial Driver, 386
sdInit
 Serial Driver, 384
sdObjectInit
 Serial Driver, 385
sdPut
 Serial Driver, 390
sdPutTimeout
 Serial Driver, 390
sdPutWouldBlock
 Serial Driver, 390
sdRead
 Serial Driver, 392
sdReadTimeout
 Serial Driver, 392
sdRequestData
 Serial Driver, 387
sdStart
 Serial Driver, 385
sdstate_t
 Serial Driver, 394
sdStop
 Serial Driver, 386
SDU_READY
 Serial over USB Driver, 401
SDU_STOP
 Serial over USB Driver, 401
SDU_UNINIT
 Serial over USB Driver, 401
sduConfigureHookI
 Serial over USB Driver, 398
sduDataReceived
 Serial over USB Driver, 399
sduDataTransmitted
 Serial over USB Driver, 399
sduInit
 Serial over USB Driver, 396
sduInterruptTransmitted
 Serial over USB Driver, 400
sduObjectInit
 Serial over USB Driver, 396
sduRequestsHook
 Serial over USB Driver, 398
sduStart
 Serial over USB Driver, 397
sdustate_t
 Serial over USB Driver, 401
sduStop
 Serial over USB Driver, 397
sdWrite
 Serial Driver, 391
sdWriteTimeout
 Serial Driver, 392
se_state
 ch_swc_event_t, 551
se_time
 ch_swc_event_t, 551
se_tp
 ch_swc_event_t, 551
se_wtobjp
 ch_swc_event_t, 551
sem
 chibios_rt::CounterSemaphore, 572
Semaphore, 670
 Counting Semaphores, 100
 s_cnt, 672
 s_queue, 672
SEMAPHORE_DECL
 Counting Semaphores, 99
sendMessage
 chibios_rt::ThreadReference, 704
Serial Driver, 381
 _serial_driver_data, 393
 _serial_driver_methods, 389
PLATFORM_SERIAL_USE_SD1, 393
SD1, 389
SD_READY, 394
SD_STOP, 394
SD_UNINIT, 394
SD_BREAK_DETECTED, 389

SD_FRAMING_ERROR, 389
sd_lld_init, 388
sd_lld_start, 388
sd_lld_stop, 388
SD_NOISE_ERROR, 389
SD_OVERRUN_ERROR, 389
SD_PARITY_ERROR, 389
sdAsynchronousRead, 393
sdAsynchronousWrite, 392
sdGet, 391
sdGetTimeout, 391
sdGetWouldBlock, 390
sdIncomingData, 386
sdInit, 384
sdObjectInit, 385
sdPut, 390
sdPutTimeout, 390
sdPutWouldBlock, 390
sdRead, 392
sdReadTimeout, 392
sdRequestData, 387
sdStart, 385
sdstate_t, 394
sdStop, 386
sdWrite, 391
sdWriteTimeout, 392
SERIAL_BUFFERS_SIZE, 389
SERIAL_DEFAULT_BITRATE, 389
SerialDriver, 394
Serial over USB Driver, 394
 _serial_usb_driver_data, 400
 _serial_usb_driver_methods, 401
SDU_READY, 401
SDU_STOP, 401
SDU_UNINIT, 401
sduConfigureHookI, 398
sduDataReceived, 399
sduDataTransmitted, 399
sdulinit, 396
sdulInterruptTransmitted, 400
sduObjectInit, 396
sduRequestsHook, 398
sduStart, 397
sdustate_t, 401
sduStop, 397
SERIAL_USB_BUFFERS_SIZE, 400
 SerialUSBDriver, 401
serial.c, 825
serial.h, 825
SERIAL_BUFFERS_SIZE
 Configuration, 478
 Serial Driver, 389
SERIAL_DEFAULT_BITRATE
 Configuration, 478
 Serial Driver, 389
serial_lld.c, 827
serial_lld.h, 827
serial_usb.c, 828
serial_usb.h, 829
SERIAL_USB_BUFFERS_SIZE
 Configuration, 478
 Serial over USB Driver, 400
SerialConfig, 672
 sc_speed, 672
SerialDriver, 672
 Serial Driver, 394
 vmt, 674
SerialDriverVMT, 674
SerialUSBConfig, 676
 bulk_in, 677
 bulk_out, 677
 int_in, 677
 usbp, 677
SerialUSBDriver, 677
 Serial over USB Driver, 401
 vmt, 679
SerialUSBDriverVMT, 679
setcurrp
 Scheduler, 67
setl
 chibios_rt::Timer, 715
setName
 chibios_rt::BaseThread, 527
setPriority
 chibios_rt::BaseThread, 527
setup
 testcase, 690
 USBDriver, 724
setup_cb
 USBEndpointConfig, 726
SETUP_CONTEXT
 Port, 204
shell.c, 830
shell.h, 831
SHELL_MAX_ARGUMENTS
 Command Shell, 485
SHELL_MAX_LINE_LENGTH
 Command Shell, 485
shell_terminated
 Command Shell, 485
shellcmd_t
 Command Shell, 486
ShellCommand, 681
 sc_function, 681
 sc_name, 681
ShellConfig, 682
 sc_channel, 682
 sc_commands, 682
shellCreate
 Command Shell, 483
shellCreateStatic
 Command Shell, 484
shellExit
 Command Shell, 483
shellGetLine
 Command Shell, 485
shellInit
 Command Shell, 483

shouldTerminate
 chibios_rt::BaseThread, 529

SID
 CANRxFrame, 548
 CANTxFrame, 549

signal
 chibios_rt::BinarySemaphore, 543
 chibios_rt::CondVar, 558
 chibios_rt::CounterSemaphore, 570

signalEvents
 chibios_rt::ThreadReference, 706

signalEventsI
 chibios_rt::ThreadReference, 706

signall
 chibios_rt::BinarySemaphore, 543
 chibios_rt::CondVar, 558
 chibios_rt::CounterSemaphore, 570

signalWait
 chibios_rt::CounterSemaphore, 571

size
 heap_header, 590
 MACReceiveDescriptor, 608
 MACTransmitDescriptor, 609

sleep
 chibios_rt::BaseThread, 530

sleep_event
 CANDriver, 547

sleepUntil
 chibios_rt::BaseThread, 530

sof_cb
 USBConfig, 721

SPI Driver, 401
 _spi_isr_code, 419
 _spi_wait_s, 418
 _spi_wakeup_isr, 419
 PLATFORM_SPI_USE_SPI1, 420
 SPI_ACTIVE, 420
 SPI_COMPLETE, 420
 SPI_READY, 420
 SPI_STOP, 420
 SPI_UNINIT, 420
 spi_lld_exchange, 413
 spi_lld_ignore, 413
 spi_lld_init, 411
 spi_lld_polled_exchange, 414
 spi_lld_receive, 414
 spi_lld_select, 412
 spi_lld_send, 413
 spi_lld_start, 412
 spi_lld_stop, 412
 spi_lld_unselect, 412
 SPI_USE_MUTUAL_EXCLUSION, 415
 SPI_USE_WAIT, 415
 spiAcquireBus, 410
 spicallback_t, 420
 SPID1, 415
 SPIDriver, 420
 spiExchange, 409
 spilgnore, 409
 spiInit, 405
 spiObjectInit, 405
 spiPolledExchange, 418
 spiReceive, 410
 spiReleaseBus, 411
 spiSelect, 406
 spiSelectl, 415
 spiSend, 409
 spiStart, 405
 spiStartExchange, 407
 spiStartExchangel, 416
 spiStartIgnore, 407
 spiStartIgnorel, 416
 spiStartReceive, 408
 spiStartReceiveI, 417
 spiStartSend, 408
 spiStartSendI, 417
 spistate_t, 420
 spiStop, 406
 spiUnselect, 407
 spiUnselectl, 415
 spi.c, 832
 spi.h, 833
 SPI_ACTIVE
 SPI Driver, 420
 SPI_COMPLETE
 SPI Driver, 420
 SPI_READY
 SPI Driver, 420
 SPI_STOP
 SPI Driver, 420
 SPI_UNINIT
 SPI Driver, 420
 spi_lld.c, 834
 spi_lld.h, 835
 spi_lld_exchange
 SPI Driver, 413
 spi_lld_ignore
 SPI Driver, 413
 spi_lld_init
 SPI Driver, 411
 spi_lld_polled_exchange
 SPI Driver, 414
 spi_lld_receive
 SPI Driver, 414
 spi_lld_select
 SPI Driver, 412
 spi_lld_send
 SPI Driver, 413
 spi_lld_start
 SPI Driver, 412
 spi_lld_stop
 SPI Driver, 412
 spi_lld_unselect
 SPI Driver, 412
 SPI_USE_MUTUAL_EXCLUSION
 Configuration, 478
 SPI Driver, 415
 SPI_USE_WAIT

Configuration, 478
SPI Driver, 415
spiAcquireBus
 SPI Driver, 410
spicallback_t
 SPI Driver, 420
SPICConfig, 683
 end_cb, 685
SPID1
 SPI Driver, 415
SPIDriver, 685
 config, 687
 mutex, 687
 SPI Driver, 420
 state, 687
 thread, 687
spiExchange
 SPI Driver, 409
spilgnore
 SPI Driver, 409
spilnit
 SPI Driver, 405
spiObjectInit
 SPI Driver, 405
spip
 MMCConfig, 634
spiPolledExchange
 SPI Driver, 418
spiReceive
 SPI Driver, 410
spiReleaseBus
 SPI Driver, 411
spiSelect
 SPI Driver, 406
spiSelectl
 SPI Driver, 415
spiSend
 SPI Driver, 409
spiStart
 SPI Driver, 405
spiStartExchange
 SPI Driver, 407
spiStartExchangel
 SPI Driver, 416
spiStartIgnore
 SPI Driver, 407
spiStartIgnorel
 SPI Driver, 416
spiStartReceive
 SPI Driver, 408
spiStartReceiveI
 SPI Driver, 417
spiStartSend
 SPI Driver, 408
spiStartSendl
 SPI Driver, 417
spistate_t
 SPI Driver, 420
spiStop

SPI Driver, 406
spiUnselect
 SPI Driver, 407
spiUnselectl
 SPI Driver, 415
STACK_ALIGN
 Port, 204
start
 chibios_rt::BaseStaticThread, 523
 chibios_rt::BaseThread, 527
 TimeMeasurement, 713
state
 ADCDriver, 504
 CANDriver, 546
 EXTDriver, 584
 GPTDriver, 589
 I2CDriver, 593
 ICUDriver, 596
 MACDriver, 607
 PWMDriver, 663
 SPIDriver, 687
 UARTDriver, 719
 USBDriver, 723
status
 USBDriver, 724
stkalign_t
 Port, 205
STM32_GPT_USE_TIM1
 GPT Driver, 253
stop
 chibios_rt::ThreadReference, 700
 TimeMeasurement, 713
Streams and Files, 178
suspend
 chibios_rt::ThreadReference, 701
suspends
 chibios_rt::ThreadReference, 701
Synchronization, 90
Synchronous Messages, 133
 chMsgGet, 136
 chMsgIsPendingI, 136
 chMsgRelease, 136
 chMsgReleaseS, 136
 chMsgSend, 134
 chMsgWait, 135
System formatted print, 490
 CHPRINTF_USE_FLOAT, 491
 chsnprintf, 490
 chvprintf, 490
System Management, 51
 _idle_thread, 54
 CH_FAST_IRQ_HANDLER, 58
 CH_IRQ_EPILOGUE, 58
 CH_IRQ_HANDLER, 58
 CH_IRQ_PROLOGUE, 58
 chSysDisable, 55
 chSysEnable, 56
 chSysGetIdleThread, 54
 chSysHalt, 55

chSysInit, 53
chSysLock, 56
chSysLockFromIlsr, 57
chSysSuspend, 56
chSysSwitch, 55
chSysTimerHandlerI, 53
chSysUnlock, 57
chSysUnlockFromIlsr, 57
WORKING_AREA, 54
SYSTEM_HALT_HOOK
 Configuration, 48
SYSTEM_TICK_EVENT_HOOK
 Configuration, 48
systime_t
 Types, 51
tb_buffer
 ch_trace_buffer_t, 553
tb_ptr
 ch_trace_buffer_t, 553
tb_size
 ch_trace_buffer_t, 553
tdsem
 MACDriver, 608
teardown
 testcase, 691
Test Runtime, 492
 DELAY_BETWEEN_TESTS, 496
 test_assert, 496
 test_assert_lock, 496
 test_assert_sequence, 497
 test_assert_time_window, 497
 test_cpu_pulse, 494
 test_emit_token, 493
 test_fail, 496
 TEST_NO_BENCHMARKS, 496
 test_print, 493
 test_println, 493
 test_printn, 493
 test_start_timer, 495
 test_terminate_threads, 493
 test_timer_done, 495
 test_wait_threads, 494
 test_wait_tick, 494
 TestThread, 495
test.c, 836
test.h, 837
test_assert
 Test Runtime, 496
test_assert_lock
 Test Runtime, 496
test_assert_sequence
 Test Runtime, 497
test_assert_time_window
 Test Runtime, 497
test_cpu_pulse
 Test Runtime, 494
test_emit_token
 Test Runtime, 493
test_fail
 Test Runtime, 496
test_print
 Test Runtime, 493
test_println
 Test Runtime, 493
test_printn
 Test Runtime, 493
test_start_timer
 Test Runtime, 495
test_terminate_threads
 Test Runtime, 493
test_timer_done
 Test Runtime, 495
test_wait_threads
 Test Runtime, 494
test_wait_tick
 Test Runtime, 494
testbmk.c, 838
 patternbmk, 838
testbmk.h, 838
 patternbmk, 839
testcase, 690
 execute, 691
 name, 690
 setup, 690
 teardown, 691
testdyn.c, 839
 patterndyn, 839
testdyn.h, 839
 patterndyn, 840
testevt.c, 840
 patternevt, 840
testevt.h, 840
 patternevt, 840
testheap.c, 841
 patternheap, 841
testheap.h, 841
 patternheap, 841
testmbox.c, 841
 patternmbox, 842
testmbox.h, 842
 patternmbox, 842
testmsg.c, 842
 patternmsg, 843
testmsg.h, 843
 patternmsg, 843
testmtx.c, 843
 patternmtx, 843
testmtx.h, 844
 patternmtx, 844
testpools.c, 844
testpools.h, 844
testqueues.c, 844
 patternqueues, 845
testqueues.h, 845
 patternqueues, 845

testsem.c, 845
 patternsem, 846
testsem.h, 846
 patternsem, 846
testthd.c, 846
 patternhd, 847
testthd.h, 847
 patternhd, 847
TestThread
 Test Runtime, 495
tfunc_t
 Threads, 83
THD_MEM_MODE_HEAP
 Threads, 80
THD_MEM_MODE_MASK
 Threads, 80
THD_MEM_MODE_MEMPOOL
 Threads, 80
THD_MEM_MODE_STATIC
 Threads, 80
THD_STATE_CURRENT
 Threads, 78
THD_STATE_FINAL
 Threads, 79
THD_STATE_NAMES
 Threads, 79
THD_STATE_READY
 Threads, 78
THD_STATE_SLEEPING
 Threads, 79
THD_STATE SNDMSG
 Threads, 79
THD_STATE SNDMSGQ
 Threads, 79
THD_STATE_SUSPENDED
 Threads, 78
THD_STATE_WTANDEVT
 Threads, 79
THD_STATE_WTCOND
 Threads, 79
THD_STATE_WTEXIT
 Threads, 79
THD_STATE_WTMSG
 Threads, 79
THD_STATE_WTMTX
 Threads, 78
THD_STATE_WTOREVT
 Threads, 79
THD_STATE_WTQUEUE
 Threads, 79
THD_STATE_WTSEM
 Threads, 78
THD_TERMINATE
 Threads, 80
THD_WA_SIZE
 Port, 204
Thread, 691
 ewmask, 696
 exitcode, 696
p_ctx, 695
p_epending, 697
p_flags, 695
p_mpool, 697
p_msg, 697
p_msgqueue, 696
p_mtxlist, 697
p_name, 695
p_newer, 695
p_next, 695
p_older, 695
p_preempt, 695
p_prev, 695
p_prio, 695
p_realprio, 697
p_refs, 695
p_state, 695
p_stklimit, 695
p_time, 696
p_waiting, 696
rdymsg, 696
wtobjp, 696
thread
 ADCDriver, 504
 SPIDriver, 687
THREAD_CONTEXT_SWITCH_HOOK
 Configuration, 48
THREAD_EXIT_HOOK
 Configuration, 48
THREAD_EXT_FIELDS
 Configuration, 47
THREAD_INIT_HOOK
 Configuration, 47
thread_ref
 chibios_rt::ThreadReference, 707
ThreadReference
 chibios_rt::ThreadReference, 700
Threads, 69
 _thread_init, 71
 _thread_memfill, 72
 chThdCreateI, 72
 chThdCreateStatic, 73
 chThdExit, 76
 chThdExitS, 76
 chThdGetPriority, 80
 chThdGetTicks, 80
 chThdLS, 81
 chThdResume, 74
 chThdResumel, 82
 chThdSelf, 80
 chThdSetPriority, 73
 chThdShouldTerminate, 81
 chThdSleep, 75
 chThdSleepMicroseconds, 83
 chThdSleepMilliseconds, 82
 chThdSleepS, 82
 chThdSleepSeconds, 82
 chThdSleepUntil, 75
 chThdTerminate, 75

chThdTerminated, 81
chThdWait, 77
chThdYield, 76
tfunc_t, 83
THD_MEM_MODE_HEAP, 80
THD_MEM_MODE_MASK, 80
THD_MEM_MODE_MEMPOOL, 80
THD_MEM_MODE_STATIC, 80
THD_STATE_CURRENT, 78
THD_STATE_FINAL, 79
THD_STATE_NAMES, 79
THD_STATE_READY, 78
THD_STATE_SLEEPING, 79
THD_STATE SNDMSG, 79
THD_STATE SNDMSGQ, 79
THD_STATE_SUSPENDED, 78
THD_STATE_WTANDEVT, 79
THD_STATE_WTCOND, 79
THD_STATE_WTEXIT, 79
THD_STATE_WTMSG, 79
THD_STATE_WTMTX, 78
THD_STATE_WTOREVT, 79
THD_STATE_WTQUEUE, 79
THD_STATE_WTSEM, 78
THD_TERMINATE, 80
ThreadsList, 707
p_next, 710
ThreadsQueue, 710
p_next, 712
p_prev, 713
THREADSQUEUE_DECL
Internals, 199
Time and Virtual Timers, 83
_vt_init, 84
chTimeElapsedSince, 89
chTimelsWithin, 89
chTimeNow, 89
chVTDOTickl, 87
chVTIsArmedl, 88
chVTReset, 88
chVTResetl, 85
chVTSet, 88
chVTSetl, 84
MS2ST, 86
S2ST, 86
US2ST, 87
VirtualTimer, 90
vfunc_t, 90
vtlist, 86
Time Measurement Driver, 420
TimeMeasurement, 422
tmInit, 421
tmObjectInit, 421
tmStartMeasurement, 422
tmStopMeasurement, 422
TIME_IMMEDIATE
Scheduler, 67
TIME_INFINITE
Scheduler, 67
TimeMeasurement, 713
best, 713
last, 713
start, 713
stop, 713
Time Measurement Driver, 422
worst, 713
timer_ref
chibios_rt::Timer, 716
tm.c, 847
tm.h, 847
tmlInit
Time Measurement Driver, 421
tmObjectInit
Time Measurement Driver, 421
tmode_t
Types, 50
tmStartMeasurement
Time Measurement Driver, 422
tmStopMeasurement
Time Measurement Driver, 422
tprio_t
Types, 50
transmitting
USBDriver, 723
trefs_t
Types, 50
TRUE
Version Numbers and Identification, 39
tryLock
chibios_rt::Mutex, 644
tryLockS
chibios_rt::Mutex, 645
tstate_t
Types, 50
txbuf
USBInEndpointState, 729
txcnt
USBInEndpointState, 729
txempty_event
CANDriver, 546
txend1_cb
UARTConfig, 717
txend2_cb
UARTConfig, 717
txqueue
USBInEndpointState, 729
txqueued
USBInEndpointState, 729
txsem
CANDriver, 546
txsize
USBInEndpointState, 729
txstate
UARTDriver, 719
Types, 49
bool_t, 50
cnt_t, 51
eventid_t, 51

eventmask_t, 51
INLINE, 50
msg_t, 50
PACK_STRUCT_BEGIN, 50
PACK_STRUCT_END, 50
PACK_STRUCT_STRUCT, 50
ROMCONST, 50
systime_t, 51
tmode_t, 50
tprio_t, 50
trefs_t, 50
tstate_t, 50

uarttxstate_t, 437
uart.c, 848
uart.h, 849
UART_READY
 UART Driver, 437
UART_RX_ACTIVE
 UART Driver, 437
UART_RX_COMPLETE
 UART Driver, 437
UART_RX_IDLE
 UART Driver, 437
UART_STOP
 UART Driver, 437
UART_TX_ACTIVE
 UART Driver, 437
UART_TX_COMPLETE
 UART Driver, 437
UART_TX_IDLE
 UART Driver, 437
UART_UNINIT
 UART Driver, 437
UART_BREAK_DETECTED
 UART Driver, 436
UART_FRAMING_ERROR
 UART Driver, 435
uart_lld_init, 433
uart_lld_start, 433
uart_lld_start_receive, 434
uart_lld_start_send, 434
uart_lld_stop, 434
uart_lld_stop_receive, 435
uart_lld_stop_send, 434
UART_NO_ERROR, 435
UART_NOISE_ERROR, 436
UART_OVERRUN_ERROR, 435
UART_PARITY_ERROR, 435
uartcb_t, 436
uartccb_t, 436
UARTD1, 435
UARTDriver, 436
uartccb_t, 436
uartflags_t, 436
uartInit, 426
uartObjectInit, 427
uartrxstate_t, 437
uartStart, 427
uartStartReceive, 430
uartStartReceive1, 431
uartStartSend, 428
uartStartSend1, 428
uartstate_t, 437
uartStop, 427
uartStopReceive, 432
uartStopReceive1, 432
uartStopSend, 429
uartStopSend1, 430

 rxchar_cb, 718
 rxend_cb, 717
 rxerr_cb, 718
 txend1_cb, 717
 txend2_cb, 717

UARTD1
 UART Driver, 435

UARTDriver, 718
 config, 719
 rxstate, 719
 state, 719
 txstate, 719
 UART Driver, 436

uartecb_t
 UART Driver, 436

uartflags_t
 UART Driver, 436

uartInit
 UART Driver, 426

uartObjectInit
 UART Driver, 427

uartrxstate_t
 UART Driver, 437

uartStart
 UART Driver, 427

uartStartReceive
 UART Driver, 430

uartStartReceive1
 UART Driver, 431

uartStartSend
 UART Driver, 428

uartStartSend1
 UART Driver, 428

uartstate_t
 UART Driver, 437

uartStop
 UART Driver, 427

uartStopReceive
 UART Driver, 432

uartStopReceive1
 UART Driver, 432

uartStopSend
 UART Driver, 429

uartStopSend1
 UART Driver, 430

uarttxstate_t
 UART Driver, 437

ud_size
 USBDescriptor, 721

ud_string
 USBDescriptor, 721

unlock
 chibios_rt::System, 689

unlockAllMutexes
 chibios_rt::BaseThread, 538

unlockFromIsr
 chibios_rt::System, 689

unlockMutex
 chibios_rt::BaseThread, 537

unlockMutexS
 chibios_rt::BaseThread, 537

unregister
 chibios_rt::EvtSource, 579

US2RTT
 HAL Driver, 260

US2ST
 Time and Virtual Timers, 87

USB Driver, 437
 _usb_ep0in, 453
 _usb_ep0out, 454
 _usb_ep0setup, 452
 _usb_isr_invoke_event_cb, 466
 _usb_isr_invoke_in_cb, 467
 _usb_isr_invoke_out_cb, 467
 _usb_isr_invoke_setup_cb, 466
 _usb_isr_invoke_sof_cb, 466
 _usb_reset, 452
 EP_STATUS_ACTIVE, 470
 EP_STATUS_DISABLED, 470
 EP_STATUS_STALLED, 470
 in, 460
 out, 460
 PLATFORM_USB_USE_USB1, 468
 USB_ACTIVE, 470
 USB_EP0_ERROR, 470
 USB_EP0_RX, 470
 USB_EP0_SENDING_STS, 470
 USB_EP0_TX, 470
 USB_EP0_WAITING_SETUP, 470
 USB_EP0_WAITING_STS, 470
 USB_EP0_WAITING_TX0, 470
 USB_EVENT_ADDRESS, 470
 USB_EVENT_CONFIGURED, 470
 USB_EVENT_RESET, 470
 USB_EVENT_STALLED, 470
 USB_EVENT_SUSPEND, 470
 USB_EVENT_WAKEUP, 470
 USB_READY, 470
 USB_SELECTED, 470
 USB_STOP, 470
 USB_UNINIT, 470
 USB_DESC_BCD, 460
 USB_DESC_BYTEx, 460
 USB_DESC_CONFIGURATION, 461
 USB_DESC_DEVICE, 461
 USB_DESC_ENDPOINT, 462
 USB_DESC_INDEX, 460
 USB_DESC_INTERFACE, 461
 USB_DESC_INTERFACE_ASSOCIATION, 461
 USB_DESC_WORD, 460
 USB_EP0_STATUS_STAGE, 467
 USB_EP_MODE_LINEAR_BUFFER, 462
 USB_EP_MODE_QUEUE_BUFFER, 462
 USB_EP_MODE_TYPE, 462
 USB_EP_MODE_TYPE_BULK, 462
 USB_EP_MODE_TYPE_CTRL, 462
 USB_EP_MODE_TYPE_INTR, 462
 USB_EP_MODE_TYPE_ISOC, 462
 usb_lld_clear_in, 460
 usb_lld_clear_out, 459
 usb_lld_connect_bus, 468
 usb_lld_disable_endpoints, 457
 usb_lld_disconnect_bus, 468

usb_lld_get_status_in, 457
usb_lld_get_status_out, 457
usb_lld_get_transaction_size, 468
usb_lld_init, 455
usb_lld_init_endpoint, 456
usb_lld_prepare_receive, 458
usb_lld_prepare_transmit, 458
usb_lld_read_setup, 458
usb_lld_reset, 456
usb_lld_set_address, 456
usb_lld_stall_in, 459
usb_lld_stall_out, 459
usb_lld_start, 455
usb_lld_start_in, 459
usb_lld_start_out, 458
usb_lld_stop, 455
USB_MAX_ENDPOINTS, 467
USB_SET_ADDRESS_ACK_HANDLING, 467
USB_SET_ADDRESS_MODE, 467
usbcallback_t, 469
usbConnectBus, 463
USBD1, 460
usbDisableEndpointsI, 446
usbDisconnectBus, 463
USBDriver, 468
usbep0state_t, 470
usbep_t, 468
usbepcallback_t, 469
usbepstatus_t, 470
usbevent_t, 470
usbeventcb_t, 469
usbFetchWord, 463
usbgetdescriptor_t, 469
usbGetDriverStatel, 462
usbGetFrameNumber, 463
usbGetReceiveStatusI, 464
usbGetReceiveTransactionSizeI, 464
usbGetTransmitStatusI, 464
usbInit, 444
usbInitEndpointI, 445
usbObjectInit, 444
usbPrepareQueuedReceive, 448
usbPrepareQueuedTransmit, 449
usbPrepareReceive, 447
usbPrepareTransmit, 447
usbReadSetup, 465
usbreqhandler_t, 469
usbSetupTransfer, 465
usbStallReceivel, 451
usbStallTransmitI, 451
usbStart, 445
usbStartReceivel, 449
usbStartTransmitI, 450
usbstate_t, 470
usbStop, 445
usb.c, 852
usb.h, 853
USB_ACTIVE
 USB Driver, 470
USB_EP0_ERROR
 USB Driver, 470
USB_EP0_RX
 USB Driver, 470
USB_EP0_SENDING_STS
 USB Driver, 470
USB_EP0_TX
 USB Driver, 470
USB_EP0_WAITING_SETUP
 USB Driver, 470
USB_EP0_WAITING_STS
 USB Driver, 470
USB_EP0_WAITING_TX0
 USB Driver, 470
USB_EVENT_ADDRESS
 USB Driver, 470
USB_EVENT_CONFIGURED
 USB Driver, 470
USB_EVENT_RESET
 USB Driver, 470
USB_EVENT_STALLED
 USB Driver, 470
USB_EVENT_SUSPEND
 USB Driver, 470
USB_EVENT_WAKEUP
 USB Driver, 470
USB_READY
 USB Driver, 470
USB_SELECTED
 USB Driver, 470
USB_STOP
 USB Driver, 470
USB_UNINIT
 USB Driver, 470
USB_DESC_BCD
 USB Driver, 460
USB_DESC_BYTE
 USB Driver, 460
USB_DESC_CONFIGURATION
 USB Driver, 461
USB_DESC_DEVICE
 USB Driver, 461
USB_DESC_ENDPOINT
 USB Driver, 462
USB_DESC_INDEX
 USB Driver, 460
USB_DESC_INTERFACE
 USB Driver, 461
USB_DESC_INTERFACE_ASSOCIATION
 USB Driver, 461
USB_DESC_WORD
 USB Driver, 460
USB_EP0_STATUS_STAGE
 USB Driver, 467
USB_EP_MODE_LINEAR_BUFFER
 USB Driver, 462
USB_EP_MODE_QUEUE_BUFFER
 USB Driver, 462
USB_EP_MODE_TYPE

USB Driver, 462
USB_EP_MODE_TYPE_BULK
 USB Driver, 462
USB_EP_MODE_TYPE_CTRL
 USB Driver, 462
USB_EP_MODE_TYPE_INTR
 USB Driver, 462
USB_EP_MODE_TYPE_ISOC
 USB Driver, 462
usb_lld.c, 856
 in, 857
 out, 857
usb_lld.h, 857
usb_lld_clear_in
 USB Driver, 460
usb_lld_clear_out
 USB Driver, 459
usb_lld_connect_bus
 USB Driver, 468
usb_lld_disable_endpoints
 USB Driver, 457
usb_lld_disconnect_bus
 USB Driver, 468
usb_lld_get_status_in
 USB Driver, 457
usb_lld_get_status_out
 USB Driver, 457
usb_lld_get_transaction_size
 USB Driver, 468
usb_lld_init
 USB Driver, 455
usb_lld_init_endpoint
 USB Driver, 456
usb_lld_prepare_receive
 USB Driver, 458
usb_lld_prepare_transmit
 USB Driver, 458
usb_lld_read_setup
 USB Driver, 458
usb_lld_reset
 USB Driver, 456
usb_lld_set_address
 USB Driver, 456
usb_lld_stall_in
 USB Driver, 459
usb_lld_stall_out
 USB Driver, 459
usb_lld_start
 USB Driver, 455
usb_lld_start_in
 USB Driver, 459
usb_lld_start_out
 USB Driver, 458
usb_lld_stop
 USB Driver, 455
USB_MAX_ENDPOINTS
 USB Driver, 467
USB_SET_ADDRESS_ACK_HANDLING
 USB Driver, 467
USB_SET_ADDRESS_MODE
 USB Driver, 467
usbcallback_t
 USB Driver, 469
USBConfig, 719
 event_cb, 721
 get_descriptor_cb, 721
 requests_hook_cb, 721
 sof_cb, 721
usbConnectBus
 USB Driver, 463
USBD1
 USB Driver, 460
USBDescriptor, 721
 ud_size, 721
 ud_string, 721
usbDisableEndpointsI
 USB Driver, 446
usbDisconnectBus
 USB Driver, 463
USBDriver, 722
 address, 724
 config, 723
 configuration, 724
 ep0endcb, 724
 ep0n, 724
 ep0next, 724
 ep0state, 723
 epc, 723
 in_params, 723
 out_params, 723
 receiving, 723
 setup, 724
 state, 723
 status, 724
 transmitting, 723
 USB Driver, 468
USBEndpointConfig, 724
 ep_mode, 726
 in_cb, 726
 in_maxsize, 726
 in_state, 727
 out_cb, 726
 out_maxsize, 726
 out_state, 727
 setup_cb, 726
usbep0state_t
 USB Driver, 470
usbep_t
 USB Driver, 468
usbepcallback_t
 USB Driver, 469
usbepstatus_t
 USB Driver, 470
usbevent_t
 USB Driver, 470
usbeventcb_t
 USB Driver, 469
usbFetchWord

USB Driver, 463
usbgetdescriptor_t
 USB Driver, 469
usbGetDriverStateI
 USB Driver, 462
usbGetFrameNumber
 USB Driver, 463
usbGetReceiveStatusI
 USB Driver, 464
usbGetReceiveTransactionSizeI
 USB Driver, 464
usbGetTransmitStatusI
 USB Driver, 464
USBInEndpointState, 727
 txbuf, 729
 txcnt, 729
 txqueue, 729
 txqueued, 729
 txsize, 729
usblInit
 USB Driver, 444
usblInitEndpointI
 USB Driver, 445
usbObjectInit
 USB Driver, 444
USBOutEndpointState, 729
 rxbuf, 731
 rcnt, 731
 rxqueue, 731
 rxqueued, 731
 rsize, 731
usbp
 SerialUSBConfig, 677
usbPrepareQueuedReceive
 USB Driver, 448
usbPrepareQueuedTransmit
 USB Driver, 449
usbPrepareReceive
 USB Driver, 447
usbPrepareTransmit
 USB Driver, 447
usbReadSetup
 USB Driver, 465
usbreqhandler_t
 USB Driver, 469
usbSetupTransfer
 USB Driver, 465
usbStallReceiveI
 USB Driver, 451
usbStallTransmitI
 USB Driver, 451
usbStart
 USB Driver, 445
usbStartReceiveI
 USB Driver, 449
usbStartTransmitI
 USB Driver, 450
usbstate_t
 USB Driver, 470

usbStop
 USB Driver, 445
Various, 479
Version Numbers and Identification, 37
 _CHIBIOS_RT_, 38
 _idle_thread, 38
 CH_FAILED, 39
 CH_KERNEL_MAJOR, 39
 CH_KERNEL_MINOR, 39
 CH_KERNEL_PATCH, 39
 CH_KERNEL_VERSION, 38
 CH_SUCCESS, 39
 FALSE, 39
 TRUE, 39
VirtualTimer, 731
 Time and Virtual Timers, 90
 vt_func, 733
 vt_next, 733
 vt_par, 733
 vt_prev, 733
 vt_time, 733
vmt
 BaseAsynchronousChannel, 506
 BaseBlockDevice, 510
 BaseChannel, 512
 BaseFileStream, 515
 BaseSequentialStream, 518
 MemoryStream, 632
 MMCDriver, 636
 MMCSDBlockDevice, 640
 SDCDriver, 668
 SerialDriver, 674
 SerialUSBDriver, 679
vt_func
 VirtualTimer, 733
vt_next
 VirtualTimer, 733
 VTList, 735
vt_par
 VirtualTimer, 733
vt_prev
 VirtualTimer, 733
 VTList, 735
vt_systime
 VTList, 735
vt_time
 VirtualTimer, 733
 VTList, 735
vtfunc_t
 Time and Virtual Timers, 90
VTList, 733
 vt_next, 735
 vt_prev, 735
 vt_systime, 735
 vt_time, 735
vtlist
 Time and Virtual Timers, 86

wait
 chibios_rt::BinarySemaphore, 541
 chibios_rt::CondVar, 560
 chibios_rt::CounterSemaphore, 568
 chibios_rt::ThreadReference, 703

waitForAllEvents
 chibios_rt::BaseThread, 534

waitForAllEventsTimeout
 chibios_rt::BaseThread, 535

waitForAnyEvent
 chibios_rt::BaseThread, 533

waitForAnyEventTimeout
 chibios_rt::BaseThread, 535

waitForMessage
 chibios_rt::BaseThread, 531

waitForOneEvent
 chibios_rt::BaseThread, 532

waitForOneEventTimeout
 chibios_rt::BaseThread, 534

waitForS
 chibios_rt::BinarySemaphore, 541
 chibios_rt::CondVar, 560
 chibios_rt::CounterSemaphore, 568

waitForTimeout
 chibios_rt::BinarySemaphore, 541
 chibios_rt::CondVar, 561
 chibios_rt::CounterSemaphore, 569

waitForTimeoutS
 chibios_rt::BinarySemaphore, 542
 chibios_rt::CounterSemaphore, 569

wakeup_event
 CANDriver, 547

width_cb
 ICUConfig, 594

WORKING_AREA
 Port, 204
 System Management, 54

worst
 TimeMeasurement, 713

write
 chibios_rt::BaseSequentialStreamInterface, 518

writeTimeout
 chibios_rt::OutQueue, 655

wtobjp
 Thread, 696

yield
 chibios_rt::BaseThread, 531