

ChibiOS/RT

7.0.0

Reference Manual

Sun Jun 13 2021 15:52:59

1 ChibiOS/RT	1
1.1 Copyright	1
1.2 Introduction	1
1.3 Related Documents	1
2 Kernel Concepts	3
2.1 Naming Conventions	3
2.2 API Name Suffixes	3
2.3 Interrupt Classes	4
2.4 System States	4
2.5 Scheduling	6
2.6 Thread States	7
2.7 Priority Levels	7
2.8 Thread Working Area	8
3 Module Index	9
3.1 Modules	9
4 Hierarchical Index	11
4.1 Class Hierarchy	11
5 Data Structure Index	13
5.1 Data Structures	13
6 File Index	17
6.1 File List	17
7 Module Documentation	21
7.1 Release and Licensing	21
7.1.1 Detailed Description	21
7.2 Release Information	22
7.2.1 Detailed Description	22
7.2.2 Macro Definition Documentation	22
7.2.2.1 __CHIBIOS__	22
7.2.2.2 CH_VERSION_STABLE	22
7.2.2.3 CH_VERSION	23
7.2.2.4 CH_VERSION_YEAR	23
7.2.2.5 CH_VERSION_MONTH	23
7.2.2.6 CH_VERSION_PATCH	23
7.2.2.7 CH_VERSION_NICKNAME	23
7.2.2.8 CH_VERSION_DATE	23
7.3 Customer Information	24
7.3.1 Detailed Description	24
7.3.2 Macro Definition Documentation	24

7.3.2.1 CH_CUSTOMER_ID_STRING	25
7.3.2.2 CH_CUSTOMER_ID_CODE	25
7.3.2.3 CH_CUSTOMER_LICENSE_EOS_DATE	25
7.3.2.4 CH_CUSTOMER_LICENSE_VERSION_YEAR	25
7.3.2.5 CH_CUSTOMER_LICENSE_VERSION_MONTH	25
7.3.2.6 CH_LICENSE	25
7.3.2.7 CH_CUSTOMER_LICENSE_VERSION_DATE	25
7.4 License Settings	26
7.4.1 Detailed Description	26
7.4.2 Macro Definition Documentation	26
7.4.2.1 CH_LICENSE_TYPE_STRING	27
7.4.2.2 CH_LICENSE_ID_STRING	27
7.4.2.3 CH_LICENSE_ID_CODE	27
7.4.2.4 CH_LICENSE_MODIFIABLE_CODE	27
7.4.2.5 CH_LICENSE_FEATURES	27
7.4.2.6 CH_LICENSE_MAX_DEPLOY	27
7.5 RT Kernel	28
7.5.1 Detailed Description	28
7.6 Version Numbers and Identification	29
7.6.1 Detailed Description	29
7.6.2 Macro Definition Documentation	29
7.6.2.1 __CHIBIOS_RT__	29
7.6.2.2 CH_KERNEL_STABLE	30
7.6.2.3 CH_KERNEL_VERSION	30
7.6.2.4 CH_KERNEL_MAJOR	30
7.6.2.5 CH_KERNEL_MINOR	30
7.6.2.6 CH_KERNEL_PATCH	30
7.6.2.7 FALSE	30
7.6.2.8 TRUE	30
7.7 Configuration	31
7.7.1 Detailed Description	31
7.8 Options	32
7.8.1 Detailed Description	32
7.8.2 Macro Definition Documentation	35
7.8.2.1 CH_CFG_SMP_MODE	35
7.8.2.2 CH_CFG_ST_RESOLUTION	36
7.8.2.3 CH_CFG_ST_FREQUENCY	36
7.8.2.4 CH_CFG_INTERVALS_SIZE	36
7.8.2.5 CH_CFG_TIME_TYPES_SIZE	36
7.8.2.6 CH_CFG_ST_TIMEDELTA	37
7.8.2.7 CH_CFG_TIME_QUANTUM	37
7.8.2.8 CH_CFG_NO_IDLE_THREAD	37

7.8.2.9 CH_CFG_OPTIMIZE_SPEED	37
7.8.2.10 CH_CFG_USE_TM	38
7.8.2.11 CH_CFG_USE_TIMESTAMP	38
7.8.2.12 CH_CFG_USE_REGISTRY	38
7.8.2.13 CH_CFG_USE_WAITEXIT	38
7.8.2.14 CH_CFG_USE_SEMAPHORES	39
7.8.2.15 CH_CFG_USE_SEMAPHORES_PRIORITY	39
7.8.2.16 CH_CFG_USE_MUTEXES	39
7.8.2.17 CH_CFG_USE_MUTEXES_RECURSIVE	39
7.8.2.18 CH_CFG_USE_CONDVARS	40
7.8.2.19 CH_CFG_USE_CONDVARS_TIMEOUT	40
7.8.2.20 CH_CFG_USE_EVENTS	40
7.8.2.21 CH_CFG_USE_EVENTS_TIMEOUT	41
7.8.2.22 CH_CFG_USE_MESSAGES	41
7.8.2.23 CH_CFG_USE_MESSAGES_PRIORITY	41
7.8.2.24 CH_CFG_USE_DYNAMIC	42
7.8.2.25 CH_CFG_USE_MAILBOXES	42
7.8.2.26 CH_CFG_USE_MEMCORE	42
7.8.2.27 CH_CFG_MEMCORE_SIZE	43
7.8.2.28 CH_CFG_USE_HEAP	43
7.8.2.29 CH_CFG_USE_MEMPOOLS	43
7.8.2.30 CH_CFG_USE_OBJ_FIFOS	44
7.8.2.31 CH_CFG_USE_PIPES	44
7.8.2.32 CH_CFG_USE_OBJ_CACHES	44
7.8.2.33 CH_CFG_USE_DELEGATES	44
7.8.2.34 CH_CFG_USE_JOBS	45
7.8.2.35 CH_CFG_USE_FACTORY	45
7.8.2.36 CH_CFG_FACTORY_MAX_NAMES_LENGTH	45
7.8.2.37 CH_CFG_FACTORY_OBJECTS_REGISTRY	45
7.8.2.38 CH_CFG_FACTORY_GENERIC_BUFFERS	45
7.8.2.39 CH_CFG_FACTORY_SEMAPHORES	46
7.8.2.40 CH_CFG_FACTORY_MAILBOXES	46
7.8.2.41 CH_CFG_FACTORY_OBJ_FIFOS	46
7.8.2.42 CH_CFG_FACTORY_PIPES	46
7.8.2.43 CH_DBG_STATISTICS	46
7.8.2.44 CH_DBG_SYSTEM_STATE_CHECK	47
7.8.2.45 CH_DBG_ENABLE_CHECKS	47
7.8.2.46 CH_DBG_ENABLE_ASSERTS	47
7.8.2.47 CH_DBG_TRACE_MASK	47
7.8.2.48 CH_DBG_TRACE_BUFFER_SIZE	48
7.8.2.49 CH_DBG_ENABLE_STACK_CHECK	48
7.8.2.50 CH_DBG_FILL_THREADS	48

7.8.2.51 CH_DBG_THREADS_PROFILING	49
7.8.2.52 CH_CFG_SYSTEM_EXTRA_FIELDS	49
7.8.2.53 CH_CFG_SYSTEM_INIT_HOOK	49
7.8.2.54 CH_CFG_OS_INSTANCE_EXTRA_FIELDS	49
7.8.2.55 CH_CFG_OS_INSTANCE_INIT_HOOK	49
7.8.2.56 CH_CFG_THREAD_EXTRA_FIELDS	50
7.8.2.57 CH_CFG_THREAD_INIT_HOOK	50
7.8.2.58 CH_CFG_THREAD_EXIT_HOOK	50
7.8.2.59 CH_CFG_CONTEXT_SWITCH_HOOK	51
7.8.2.60 CH_CFG_IRQ_PROLOGUE_HOOK	51
7.8.2.61 CH_CFG_IRQ_EPILOGUE_HOOK	51
7.8.2.62 CH_CFG_IDLE_ENTER_HOOK	52
7.8.2.63 CH_CFG_IDLE_LEAVE_HOOK	52
7.8.2.64 CH_CFG_IDLE_LOOP_HOOK	52
7.8.2.65 CH_CFG_SYSTEM_TICK_HOOK	53
7.8.2.66 CH_CFG_SYSTEM_HALT_HOOK	53
7.8.2.67 CH_CFG_TRACE_HOOK	53
7.8.2.68 CH_CFG_RUNTIME_FAULTS_HOOK	53
7.9 Checks	54
7.10 Restrictions	55
7.10.1 Detailed Description	55
7.11 System	56
7.11.1 Detailed Description	56
7.12 Port Interface	57
7.12.1 Detailed Description	57
7.13 OS Types and Structures	58
7.13.1 Detailed Description	58
7.13.2 Macro Definition Documentation	59
7.13.2.1 __CH_STRINGIFY	59
7.13.2.2 __CH_OFFSETOF	60
7.13.2.3 __CH_USED	60
7.13.2.4 likely	60
7.13.2.5 unlikely	61
7.13.3 Typedef Documentation	61
7.13.3.1 rtcnt_t	61
7.13.3.2 rttime_t	61
7.13.3.3 syssts_t	61
7.13.3.4 stklalign_t	62
7.13.3.5 tmode_t	62
7.13.3.6 tstate_t	62
7.13.3.7 trefs_t	62
7.13.3.8 tslices_t	62

7.13.3.9 <code>tprio_t</code>	62
7.13.3.10 <code>msg_t</code>	62
7.13.3.11 <code>eventid_t</code>	63
7.13.3.12 <code>eventmask_t</code>	63
7.13.3.13 <code>eventflags_t</code>	63
7.13.3.14 <code>cnt_t</code>	63
7.13.3.15 <code>ucnt_t</code>	63
7.13.3.16 <code>core_id_t</code>	63
7.13.3.17 <code>thread_t</code>	64
7.13.3.18 <code>os_instance_t</code>	64
7.13.3.19 <code>virtual_timer_t</code>	64
7.13.3.20 <code>vfunc_t</code>	64
7.13.3.21 <code>virtual_timers_list_t</code>	64
7.13.3.22 <code>registry_t</code>	65
7.13.3.23 <code>thread_reference_t</code>	65
7.13.3.24 <code>threads_queue_t</code>	65
7.13.3.25 <code>ready_list_t</code>	65
7.13.3.26 <code>os_instance_config_t</code>	65
7.13.3.27 <code>ch_system_t</code>	66
7.13.4 Enumeration Type Documentation	66
7.13.4.1 <code>system_state_t</code>	66
7.13.5 Function Documentation	66
7.13.5.1 <code>chSysHalt()</code>	66
7.14 OS Instances	68
7.14.1 Detailed Description	68
7.14.2 Macro Definition Documentation	68
7.14.2.1 <code>__instance_get_currthread</code>	68
7.14.2.2 <code>__instance_set_currthread</code>	68
7.14.3 Function Documentation	69
7.14.3.1 <code>__idle_thread()</code>	69
7.14.3.2 <code>chInstanceObjectInit()</code>	69
7.15 Runtime Faults Collection Unit	71
7.15.1 Detailed Description	71
7.15.2 Macro Definition Documentation	71
7.15.2.1 <code>CH_RFCU_ALLFAULTS</code>	71
7.15.3 Typedef Documentation	72
7.15.3.1 <code>rfcu_mask_t</code>	72
7.15.3.2 <code>rfcu_t</code>	72
7.15.4 Function Documentation	72
7.15.4.1 <code>chRFCUCollectFaults()</code>	72
7.15.4.2 <code>chRFCUGetAndClearFaults()</code>	72
7.15.4.3 <code>__rfcu_object_init()</code>	73

7.16 Lists and Queues	74
7.16.1 Detailed Description	74
7.16.2 Macro Definition Documentation	75
7.16.2.1 __CH_QUEUE_DATA	75
7.16.2.2 CH_QUEUE_DECL	76
7.16.3 Typedef Documentation	76
7.16.3.1 ch_list_t	76
7.16.3.2 ch_queue_t	76
7.16.3.3 ch_priority_queue_t	76
7.16.3.4 ch_delta_list_t	77
7.16.4 Function Documentation	77
7.16.4.1 ch_list_init()	77
7.16.4.2 ch_list_isempty()	77
7.16.4.3 ch_list_notempty()	78
7.16.4.4 ch_list_link()	78
7.16.4.5 ch_list_unlink()	78
7.16.4.6 ch_queue_init()	79
7.16.4.7 ch_queue_isempty()	79
7.16.4.8 ch_queue_notempty()	80
7.16.4.9 ch_queue_insert()	80
7.16.4.10 ch_queue_fifo_remove()	80
7.16.4.11 ch_queue_lifo_remove()	81
7.16.4.12 ch_queue_dequeue()	82
7.16.4.13 ch_pqueue_init()	82
7.16.4.14 ch_pqueue_remove_highest()	82
7.16.4.15 ch_pqueue_insert_behind()	83
7.16.4.16 ch_pqueue_insert_ahead()	83
7.16.4.17 ch_dlist_init()	84
7.16.4.18 ch_dlist_isempty()	84
7.16.4.19 ch_dlist_notempty()	85
7.16.4.20 ch_dlist_islast()	85
7.16.4.21 ch_dlist_isfirst()	85
7.16.4.22 ch_dlist_insert_after()	86
7.16.4.23 ch_dlist_insert_before()	86
7.16.4.24 ch_dlist_insert()	87
7.16.4.25 ch_dlist_remove_first()	87
7.16.4.26 ch_dlist_dequeue()	88
7.17 Scheduler	89
7.17.1 Detailed Description	89
7.17.2 Macro Definition Documentation	92
7.17.2.1 MSG_OK	92
7.17.2.2 MSG_TIMEOUT	92

7.17.2.3 MSG_RESET	92
7.17.2.4 NOPRIO	92
7.17.2.5 IDLEPRIO	92
7.17.2.6 LOWPRIO	93
7.17.2.7 NORMALPRIO	93
7.17.2.8 HIGHPRIO	93
7.17.2.9 CH_STATE_READY	93
7.17.2.10 CH_STATE_CURRENT	93
7.17.2.11 CH_STATE_WTSTART	93
7.17.2.12 CH_STATE_SUSPENDED	94
7.17.2.13 CH_STATE_QUEUED	94
7.17.2.14 CH_STATE_WTSEM	94
7.17.2.15 CH_STATE_WTMTX	94
7.17.2.16 CH_STATE_WTCOND	94
7.17.2.17 CH_STATE_SLEEPING	94
7.17.2.18 CH_STATE_WTEXIT	95
7.17.2.19 CH_STATE_WTOREVT	95
7.17.2.20 CH_STATE_WTANDEVT	95
7.17.2.21 CH_STATE SNDMSGQ	95
7.17.2.22 CH_STATE SNDMSG	95
7.17.2.23 CH_STATE_WTMSG	95
7.17.2.24 CH_STATE_FINAL	96
7.17.2.25 CH_STATE_NAMES	96
7.17.2.26 CH_FLAG_MODE_MASK	96
7.17.2.27 CH_FLAG_MODE_STATIC	96
7.17.2.28 CH_FLAG_MODE_HEAP	96
7.17.2.29 CH_FLAG_MODE_MPOOL	97
7.17.2.30 CH_FLAG_TERMINATE	97
7.17.2.31 firstprio	97
7.17.2.32 __sch_get_currthread	97
7.17.3 Function Documentation	97
7.17.3.1 __sch_ready_behind()	98
7.17.3.2 __sch_ready_ahead()	99
7.17.3.3 __sch_reschedule_behind()	100
7.17.3.4 __sch_reschedule_ahead()	101
7.17.3.5 ch_sch_prio_insert()	101
7.17.3.6 chSchReadyI()	102
7.17.3.7 chSchGoSleepS()	103
7.17.3.8 chSchGoSleepTimeoutS()	104
7.17.3.9 chSchWakeups()	105
7.17.3.10 chSchRescheduleS()	106
7.17.3.11 chSchIsPreemptionRequired()	107

7.17.3.12 chSchDoPreemption()	107
7.17.3.13 chSchPreemption()	108
7.17.3.14 chSchDoYieldS()	108
7.17.3.15 chSchSelectFirstI()	109
7.18 Base Kernel Services	110
7.18.1 Detailed Description	110
7.19 System Management	111
7.19.1 Detailed Description	111
7.19.2 Macro Definition Documentation	113
7.19.2.1 CH_SYS_CORE0_MEMORY	113
7.19.2.2 CH_SYS_CORE1_MEMORY	114
7.19.2.3 currcore	114
7.19.2.4 CH_IRQ_IS_VALID_PRIORITY	114
7.19.2.5 CH_IRQ_IS_VALID_KERNEL_PRIORITY	114
7.19.2.6 CH_IRQ_PROLOGUE	115
7.19.2.7 CH_IRQ_EPILOGUE	116
7.19.2.8 CH_IRQ_HANDLER	116
7.19.2.9 CH_FAST_IRQ_HANDLER	116
7.19.2.10 S2RTC	117
7.19.2.11 MS2RTC	117
7.19.2.12 US2RTC	118
7.19.2.13 RTC2S	118
7.19.2.14 RTC2MS	119
7.19.2.15 RTC2US	120
7.19.2.16 chSysGetRealtimeCounterX	120
7.19.2.17 chSysSwitch	121
7.19.3 Function Documentation	121
7.19.3.1 THD_WORKING_AREA() [1/2]	121
7.19.3.2 THD_WORKING_AREA() [2/2]	121
7.19.3.3 chSysWaitSystemState()	122
7.19.3.4 chSysInit()	122
7.19.3.5 chSysHalt()	122
7.19.3.6 chSysIntegrityCheckI()	123
7.19.3.7 chSysTimerHandlerI()	124
7.19.3.8 chSysGetStatusAndLockX()	125
7.19.3.9 chSysRestoreStatusX()	126
7.19.3.10 chSysIsCounterWithinX()	127
7.19.3.11 chSysPolledDelayX()	128
7.19.3.12 chSysDisable()	128
7.19.3.13 chSysSuspend()	129
7.19.3.14 chSysEnable()	130
7.19.3.15 chSysLock()	130

7.19.3.16 chSysUnlock()	131
7.19.3.17 chSysLockFromISR()	131
7.19.3.18 chSysUnlockFromISR()	132
7.19.3.19 chSysUnconditionalLock()	133
7.19.3.20 chSysUnconditionalUnlock()	133
7.19.3.21 chSysNotifyInstance()	134
7.19.3.22 chSysGetIdleThreadX()	134
7.19.4 Variable Documentation	134
7.19.4.1 ch_system	134
7.19.4.2 ch0	135
7.19.4.3 ch_core0_cfg	135
7.19.4.4 ch1	135
7.19.4.5 ch_core1_cfg	135
7.20 Memory Alignment	136
7.20.1 Detailed Description	136
7.20.2 Macro Definition Documentation	136
7.20.2.1 MEM_ALIGN_MASK	136
7.20.2.2 MEM_ALIGN_PREV	136
7.20.2.3 MEM_ALIGN_NEXT	137
7.20.2.4 MEM_IS_ALIGNED	137
7.20.2.5 MEM_IS_VALID_ALIGNMENT	137
7.21 Time and Intervals	139
7.21.1 Detailed Description	139
7.21.2 Macro Definition Documentation	140
7.21.2.1 TIME_IMMEDIATE	140
7.21.2.2 TIME_INFINITE	141
7.21.2.3 TIME_MAX_INTERVAL	141
7.21.2.4 TIME_MAX_SYSTIME	141
7.21.2.5 TIME_S2I	141
7.21.2.6 TIME_MS2I	142
7.21.2.7 TIME_US2I	142
7.21.2.8 TIME_I2S	143
7.21.2.9 TIME_I2MS	144
7.21.2.10 TIME_I2US	144
7.21.3 Typedef Documentation	145
7.21.3.1 systime_t	145
7.21.3.2 sysinterval_t	145
7.21.3.3 systimestamp_t	145
7.21.3.4 time_secs_t	146
7.21.3.5 time_msecs_t	146
7.21.3.6 time_usecs_t	146
7.21.3.7 time_conv_t	146

7.21.4 Function Documentation	146
7.21.4.1 chTimeS2I()	146
7.21.4.2 chTimeMS2I()	147
7.21.4.3 chTimeUS2I()	147
7.21.4.4 chTimel2S()	148
7.21.4.5 chTimel2MS()	148
7.21.4.6 chTimel2US()	149
7.21.4.7 chTimeAddX()	149
7.21.4.8 chTimeDiffX()	150
7.21.4.9 chTimelsInRangeX()	150
7.21.4.10 chTimeStampAddX()	151
7.21.4.11 chTimeStampDiffX()	151
7.21.4.12 chTimeStampIsInRangeX()	152
7.22 Virtual Timers	153
7.22.1 Detailed Description	153
7.22.2 Function Documentation	154
7.22.2.1 vt_insert_first()	154
7.22.2.2 vt_enqueue()	155
7.22.2.3 chVTDosetl()	155
7.22.2.4 chVTDoSetContinuousl()	156
7.22.2.5 chVTDoResetl()	157
7.22.2.6 chVTGetRemainingInterval()	158
7.22.2.7 chVTDotickl()	159
7.22.2.8 chVTGetTimeStapl()	160
7.22.2.9 chVTRest setTimeStapl()	161
7.22.2.10 chVTOBJECTINIT()	162
7.22.2.11 chVTGetSystemTimeX()	162
7.22.2.12 chVTGetSystemTime()	163
7.22.2.13 chVTTIMEELAPSED SINCE X()	163
7.22.2.14 chVTIsSystemTimeWithinX()	164
7.22.2.15 chVTIsSystemTimeWithin()	165
7.22.2.16 chVTGetTimersStateL()	166
7.22.2.17 chVTIsArmedL()	167
7.22.2.18 chVTIsArmed()	168
7.22.2.19 chVTResetl()	169
7.22.2.20 chVTReset()	170
7.22.2.21 chVTSetl()	171
7.22.2.22 chVTSet()	172
7.22.2.23 chVTSetContinuousl()	173
7.22.2.24 chVTSetContinuous()	174
7.22.2.25 chVTGetReloadIntervalX()	175
7.22.2.26 chVTSetReloadIntervalX()	176

7.22.2.27 chVTGetTimeStamp()	176
7.22.2.28 chVTRestartTimeStamp()	177
7.22.2.29 __vt_object_init()	178
7.23 Threads	180
7.23.1 Detailed Description	180
7.23.2 Macro Definition Documentation	183
7.23.2.1 __THREADS_QUEUE_DATA	183
7.23.2.2 THREADS_QUEUE_DECL	183
7.23.2.3 THD_WORKING_AREA_SIZE	184
7.23.2.4 THD_WORKING_AREA	184
7.23.2.5 THD_WORKING_AREA_BASE	184
7.23.2.6 THD_WORKING_AREA_END	185
7.23.2.7 THD_FUNCTION	185
7.23.2.8 THD_DESCRIPTOR	185
7.23.2.9 THD_DESCRIPTOR_AFFINITY	187
7.23.2.10 chThdSleepSeconds	187
7.23.2.11 chThdSleepMilliseconds	188
7.23.2.12 chThdSleepMicroseconds	188
7.23.3 Typedef Documentation	189
7.23.3.1 tfunc_t	189
7.23.4 Function Documentation	189
7.23.4.1 __thd_object_init()	189
7.23.4.2 __thd_memfill()	190
7.23.4.3 chThdCreateSuspended()	191
7.23.4.4 chThdCreateSuspended()	192
7.23.4.5 chThdCreate()	193
7.23.4.6 chThdCreate()	194
7.23.4.7 chThdCreateStatic()	195
7.23.4.8 chThdStart()	196
7.23.4.9 chThdAddRef()	197
7.23.4.10 chThdRelease()	198
7.23.4.11 chThdExit()	199
7.23.4.12 chThdExitS()	200
7.23.4.13 chThdWait()	201
7.23.4.14 chThdSetPriority()	202
7.23.4.15 chThdTerminate()	203
7.23.4.16 chThdSleep()	204
7.23.4.17 chThdSleepUntil()	205
7.23.4.18 chThdSleepUntilWindowed()	206
7.23.4.19 chThdYield()	207
7.23.4.20 chThdSuspendS()	208
7.23.4.21 chThdSuspendTimeoutS()	209

7.23.4.22 chThdResumel()	210
7.23.4.23 chThdResumeS()	211
7.23.4.24 chThdResume()	212
7.23.4.25 chThdEnqueueTimeoutS()	213
7.23.4.26 chThdDequeueNextl()	214
7.23.4.27 chThdDequeueAlll()	215
7.23.4.28 chThdGetSelfX()	216
7.23.4.29 chThdGetPriorityX()	216
7.23.4.30 chThdGetTicksX()	216
7.23.4.31 chThdGetWorkingAreaX()	217
7.23.4.32 chThdTerminatedX()	217
7.23.4.33 chThdShouldTerminateX()	218
7.23.4.34 chThdStartl()	218
7.23.4.35 chThdSleepS()	219
7.23.4.36 chThdQueueObjectInit()	220
7.23.4.37 chThdQueueIsEmptyl()	220
7.23.4.38 chThdDoDequeueNextl()	221
7.24 Time Measurement	223
7.24.1 Detailed Description	223
7.24.2 Macro Definition Documentation	223
7.24.2.1 TM_CALIBRATION_LOOP	223
7.24.3 Function Documentation	223
7.24.3.1 chTMObjectInit()	223
7.24.3.2 chTMStartMeasurementX()	224
7.24.3.3 chTMStopMeasurementX()	224
7.24.3.4 chTMChainMeasurementToX()	225
7.24.3.5 __tm_calibration_object_init()	225
7.25 Synchronization	227
7.25.1 Detailed Description	227
7.26 Counting Semaphores	228
7.26.1 Detailed Description	228
7.26.2 Macro Definition Documentation	229
7.26.2.1 __SEMAPHORE_DATA	229
7.26.2.2 SEMAPHORE_DECL	230
7.26.3 Typedef Documentation	230
7.26.3.1 semaphore_t	230
7.26.4 Function Documentation	230
7.26.4.1 chSemObjectInit()	230
7.26.4.2 chSemResetWithMessage()	231
7.26.4.3 chSemResetWithMessageI()	232
7.26.4.4 chSemWait()	233
7.26.4.5 chSemWaitS()	234

7.26.4.6 chSemWaitTimeout()	235
7.26.4.7 chSemWaitTimeoutS()	236
7.26.4.8 chSemSignal()	237
7.26.4.9 chSemSignall()	238
7.26.4.10 chSemAddCounterl()	239
7.26.4.11 chSemSignalWait()	240
7.26.4.12 chSemReset()	241
7.26.4.13 chSemResetl()	242
7.26.4.14 chSemFastWaitl()	243
7.26.4.15 chSemFastSignall()	244
7.26.4.16 chSemGetCounterl()	244
7.27 Mutexes	246
7.27.1 Detailed Description	246
7.27.2 Macro Definition Documentation	247
7.27.2.1 __MUTEX_DATA	248
7.27.2.2 MUTEX_DECL	249
7.27.3 Typedef Documentation	249
7.27.3.1 mutex_t	249
7.27.4 Function Documentation	249
7.27.4.1 chMtxObjectInit()	249
7.27.4.2 chMtxLock()	250
7.27.4.3 chMtxLockS()	251
7.27.4.4 chMtxTryLock()	252
7.27.4.5 chMtxTryLockS()	253
7.27.4.6 chMtxUnlock()	254
7.27.4.7 chMtxUnlockS()	255
7.27.4.8 chMtxUnlockAllS()	256
7.27.4.9 chMtxUnlockAll()	257
7.27.4.10 chMtxQueueNotEmptyS()	258
7.27.4.11 chMtxGetOwnerl()	259
7.27.4.12 chMtxGetNextMutexX()	260
7.28 Condition Variables	261
7.28.1 Detailed Description	261
7.28.2 Macro Definition Documentation	262
7.28.2.1 __CONDVAR_DATA	262
7.28.2.2 CONDVAR_DECL	262
7.28.3 Typedef Documentation	262
7.28.3.1 condition_variable_t	262
7.28.4 Function Documentation	262
7.28.4.1 chCondObjectInit()	263
7.28.4.2 chCondSignal()	263
7.28.4.3 chCondSignall()	264

7.28.4.4 chCondBroadcast()	265
7.28.4.5 chCondBroadcastI()	266
7.28.4.6 chCondWait()	267
7.28.4.7 chCondWaitS()	268
7.28.4.8 chCondWaitTimeout()	269
7.28.4.9 chCondWaitTimeoutS()	271
7.29 Event Flags	273
7.29.1 Detailed Description	273
7.29.2 Macro Definition Documentation	275
7.29.2.1 ALL_EVENTS	275
7.29.2.2 EVENT_MASK	275
7.29.2.3 __EVENTSOURCE_DATA	275
7.29.2.4 EVENTSOURCE_DECL	276
7.29.3 Typedef Documentation	276
7.29.3.1 event_source_t	276
7.29.3.2 evhandler_t	276
7.29.4 Function Documentation	276
7.29.4.1 chEvtRegisterMaskWithFlagsI()	277
7.29.4.2 chEvtRegisterMaskWithFlags()	277
7.29.4.3 chEvtUnregister()	278
7.29.4.4 chEvtGetAndClearEventsI()	279
7.29.4.5 chEvtGetAndClearEvents()	280
7.29.4.6 chEvtAddEvents()	281
7.29.4.7 chEvtGetAndClearFlagsI()	282
7.29.4.8 chEvtGetAndClearFlags()	283
7.29.4.9 chEvtSignalI()	284
7.29.4.10 chEvtSignal()	285
7.29.4.11 chEvtBroadcastFlagsI()	286
7.29.4.12 chEvtBroadcastFlags()	287
7.29.4.13 chEvtDispatch()	288
7.29.4.14 chEvtWaitOne()	288
7.29.4.15 chEvtWaitAny()	289
7.29.4.16 chEvtWaitAll()	290
7.29.4.17 chEvtWaitOneTimeout()	291
7.29.4.18 chEvtWaitAnyTimeout()	292
7.29.4.19 chEvtWaitAllTimeout()	293
7.29.4.20 chEvtObjectInit()	294
7.29.4.21 chEvtRegisterMask()	295
7.29.4.22 chEvtRegister()	295
7.29.4.23 chEvtIsListeningI()	296
7.29.4.24 chEvtBroadcast()	297
7.29.4.25 chEvtBroadcastI()	297

7.29.4.26 chEvtAddEventsI()	298
7.29.4.27 chEvtGetEventsX()	298
7.30 Synchronous Messages	300
7.30.1 Detailed Description	300
7.30.2 Function Documentation	301
7.30.2.1 chMsgSend()	301
7.30.2.2 chMsgWaitS()	301
7.30.2.3 chMsgWaitTimeoutS()	302
7.30.2.4 chMsgPollS()	303
7.30.2.5 chMsgRelease()	304
7.30.2.6 chMsgWait()	305
7.30.2.7 chMsgWaitTimeout()	306
7.30.2.8 chMsgPoll()	307
7.30.2.9 chMsgIsPendingI()	309
7.30.2.10 chMsgGet()	309
7.30.2.11 chMsgReleaseS()	310
7.31 Dynamic Threads	311
7.31.1 Detailed Description	311
7.31.2 Function Documentation	311
7.31.2.1 chThdCreateFromHeap()	311
7.31.2.2 chThdCreateFromMemoryPool()	312
7.32 Registry	314
7.32.1 Detailed Description	314
7.32.2 Macro Definition Documentation	315
7.32.2.1 REG_HEADER	315
7.32.2.2 REG_REMOVE	315
7.32.2.3 REG_INSERT	316
7.32.3 Function Documentation	316
7.32.3.1 chRegFirstThread()	316
7.32.3.2 chRegNextThread()	317
7.32.3.3 chRegFindThreadByName()	318
7.32.3.4 chRegFindThreadByPointer()	319
7.32.3.5 chRegFindThreadByWorkingArea()	320
7.32.3.6 __reg_object_init()	321
7.32.3.7 chRegSetThreadName()	322
7.32.3.8 chRegGetThreadNameX()	322
7.32.3.9 chRegSetThreadNameX()	323
7.33 Debug	324
7.33.1 Detailed Description	324
7.34 Checks and Assertions	325
7.34.1 Detailed Description	325
7.34.2 Macro Definition Documentation	327

7.34.2.1 CH_DBG_STACK_FILL_VALUE	327
7.34.2.2 chDbgCheck	327
7.34.2.3 chDbgAssert	327
7.34.3 TYPEDOC Documentation	328
7.34.3.1 system_debug_t	328
7.34.4 FD Documentation	328
7.34.4.1 __dbg_check_disable()	329
7.34.4.2 __dbg_check_suspend()	329
7.34.4.3 __dbg_check_enable()	330
7.34.4.4 __dbg_check_lock()	330
7.34.4.5 __dbg_check_unlock()	331
7.34.4.6 __dbg_check_lock_from_isr()	331
7.34.4.7 __dbg_check_unlock_from_isr()	332
7.34.4.8 __dbg_check_enter_isr()	332
7.34.4.9 __dbg_check_leave_isr()	333
7.34.4.10 chDbgCheckClassI()	333
7.34.4.11 chDbgCheckClassS()	334
7.34.4.12 __dbg_object_init()	334
7.35 Tracing	335
7.35.1 Detailed Description	335
7.35.2 Macro Definition Documentation	336
7.35.2.1 CH_DBG_TRACE_MASK	336
7.35.2.2 CH_DBG_TRACE_BUFFER_SIZE	336
7.35.3 FD Documentation	337
7.35.3.1 trace_next()	337
7.35.3.2 __trace_object_init()	337
7.35.3.3 __trace_ready()	338
7.35.3.4 __trace_switch()	338
7.35.3.5 __trace_isr_enter()	338
7.35.3.6 __trace_isr_leave()	339
7.35.3.7 __trace_halt()	339
7.35.3.8 chTraceWritel()	339
7.35.3.9 chTraceWrite()	340
7.35.3.10 chTraceSuspendI()	341
7.35.3.11 chTraceSuspend()	342
7.35.3.12 chTraceResumeI()	343
7.35.3.13 chTraceResume()	344
7.36 Statistics	345
7.36.1 Detailed Description	345
7.36.2 FD Documentation	345
7.36.2.1 __stats_increase_irq()	345
7.36.2.2 __stats_ctxtswc()	345

7.36.2.3 __stats_start_measure_crit_thd()	346
7.36.2.4 __stats_stop_measure_crit_thd()	346
7.36.2.5 __stats_start_measure_crit_isr()	347
7.36.2.6 __stats_stop_measure_crit_isr()	347
7.36.2.7 __stats_object_init()	348
7.37 OS Library	349
7.37.1 Detailed Description	349
7.38 Version Numbers and Identification	350
7.38.1 Detailed Description	350
7.38.2 Macro Definition Documentation	350
7.38.2.1 __CHIBIOS_OSLIB_	350
7.38.2.2 CH_OSLIB_STABLE	350
7.38.2.3 CH_OSLIB_VERSION	351
7.38.2.4 CH_OSLIB_MAJOR	351
7.38.2.5 CH_OSLIB_MINOR	351
7.38.2.6 CH_OSLIB_PATCH	351
7.38.3 Function Documentation	351
7.38.3.1 __oslib_init()	352
7.39 Synchronization	353
7.39.1 Detailed Description	353
7.40 Binary Semaphores	354
7.40.1 Detailed Description	354
7.40.2 Macro Definition Documentation	354
7.40.2.1 __BSEMAPHORE_DATA	355
7.40.2.2 BSEMAPHORE_DECL	355
7.40.3 Typedef Documentation	355
7.40.3.1 binary_semaphore_t	355
7.40.4 Function Documentation	355
7.40.4.1 chBSemObjectInit()	356
7.40.4.2 chBSemWait()	356
7.40.4.3 chBSemWaitS()	357
7.40.4.4 chBSemWaitTimeoutS()	358
7.40.4.5 chBSemWaitTimeout()	359
7.40.4.6 chBSemReset()	360
7.40.4.7 chBSemReset()	361
7.40.4.8 chBSemSignal()	362
7.40.4.9 chBSemSignal()	362
7.40.4.10 chBSemGetState()	363
7.41 Mailboxes	365
7.41.1 Detailed Description	365
7.41.2 Macro Definition Documentation	366
7.41.2.1 __MAILBOX_DATA	367

7.41.2.2 MAILBOX_DECL	367
7.41.3 Function Documentation	367
7.41.3.1 chMBOBJECTINIT()	368
7.41.3.2 chMBReset()	368
7.41.3.3 chMBResetl()	369
7.41.3.4 chMBPostTimeout()	370
7.41.3.5 chMBPostTimeoutS()	371
7.41.3.6 chMBPostl()	372
7.41.3.7 chMBPostAheadTimeout()	373
7.41.3.8 chMBPostAheadTimeoutS()	374
7.41.3.9 chMBPostAheadl()	375
7.41.3.10 chMBFetchTimeout()	376
7.41.3.11 chMBFetchTimeoutS()	377
7.41.3.12 chMBFetchl()	378
7.41.3.13 chMBGetSizel()	379
7.41.3.14 chMBGetUsedCountl()	380
7.41.3.15 chMBGetFreeCountl()	380
7.41.3.16 chMBPeekl()	381
7.41.3.17 chMBResumeX()	382
7.42 Pipes	383
7.42.1 Detailed Description	383
7.42.2 Macro Definition Documentation	383
7.42.2.1 __PIPE_DATA	384
7.42.2.2 PIPE_DECL	384
7.42.3 Function Documentation	384
7.42.3.1 pipe_write()	385
7.42.3.2 pipe_read()	385
7.42.3.3 chPipeObjectInit()	386
7.42.3.4 chPipeReset()	386
7.42.3.5 chPipeWriteTimeout()	387
7.42.3.6 chPipeReadTimeout()	387
7.42.3.7 chPipeGetSize()	388
7.42.3.8 chPipeGetUsedCount()	388
7.42.3.9 chPipeGetFreeCount()	389
7.42.3.10 chPipeResume()	389
7.43 Delegate Threads	391
7.43.1 Detailed Description	391
7.43.2 Typedef Documentation	392
7.43.2.1 delegate_veneer_t	392
7.43.2.2 delegate_fn0_t	392
7.43.2.3 delegate_fn1_t	392
7.43.2.4 delegate_fn2_t	392

7.43.2.5 <code>delegate_fn3_t</code>	392
7.43.2.6 <code>delegate_fn4_t</code>	392
7.43.3 Function Documentation	392
7.43.3.1 <code>__ch_delegate_fn0()</code>	392
7.43.3.2 <code>__ch_delegate_fn1()</code>	393
7.43.3.3 <code>__ch_delegate_fn2()</code>	393
7.43.3.4 <code>__ch_delegate_fn3()</code>	393
7.43.3.5 <code>__ch_delegate_fn4()</code>	395
7.43.3.6 <code>chDelegateCallVeneer()</code>	395
7.43.3.7 <code>chDelegateDispatch()</code>	396
7.43.3.8 <code>chDelegateDispatchTimeout()</code>	397
7.43.3.9 <code>chDelegateCallDirect0()</code>	398
7.43.3.10 <code>chDelegateCallDirect1()</code>	398
7.43.3.11 <code>chDelegateCallDirect2()</code>	399
7.43.3.12 <code>chDelegateCallDirect3()</code>	400
7.43.3.13 <code>chDelegateCallDirect4()</code>	401
7.44 Jobs Queues	403
7.44.1 Detailed Description	403
7.44.2 Macro Definition Documentation	404
7.44.2.1 <code>MSG_JOB_NULL</code>	404
7.44.3 Typedef Documentation	404
7.44.3.1 <code>jobs_queue_t</code>	404
7.44.3.2 <code>job_function_t</code>	404
7.44.3.3 <code>job_descriptor_t</code>	404
7.44.4 Function Documentation	404
7.44.4.1 <code>chJobObjectInit()</code>	404
7.44.4.2 <code>chJobGet()</code>	405
7.44.4.3 <code>chJobGetI()</code>	406
7.44.4.4 <code>chJobGetTimeoutS()</code>	407
7.44.4.5 <code>chJobGetTimeout()</code>	408
7.44.4.6 <code>chJobPostI()</code>	409
7.44.4.7 <code>chJobPostS()</code>	410
7.44.4.8 <code>chJobPost()</code>	411
7.44.4.9 <code>chJobPostAheadI()</code>	412
7.44.4.10 <code>chJobPostAheadS()</code>	413
7.44.4.11 <code>chJobPostAhead()</code>	414
7.44.4.12 <code>chJobDispatch()</code>	415
7.44.4.13 <code>chJobDispatchTimeout()</code>	416
7.45 Memory Management	418
7.45.1 Detailed Description	418
7.46 Core Memory Manager	419
7.46.1 Detailed Description	419

7.46.2 Macro Definition Documentation	420
7.46.2.1 CH_CFG_MEMCORE_SIZE	420
7.46.2.2 chCoreAllocAlignedWithOffset	421
7.46.2.3 chCoreAllocAlignedWithOffset	421
7.46.3 Typedef Documentation	421
7.46.3.1 memgetfunc_t	421
7.46.3.2 memgetfunc2_t	421
7.46.4 Function Documentation	422
7.46.4.1 __core_init()	422
7.46.4.2 chCoreAllocFromBaseI()	422
7.46.4.3 chCoreAllocFromTopI()	423
7.46.4.4 chCoreAllocFromBase()	424
7.46.4.5 chCoreAllocFromTop()	425
7.46.4.6 chCoreGetStatusX()	426
7.46.4.7 chCoreAllocAlignedI()	426
7.46.4.8 chCoreAllocAligned()	427
7.46.4.9 chCoreAllocI()	428
7.46.4.10 chCoreAlloc()	428
7.46.5 Variable Documentation	429
7.46.5.1 ch_memcore	429
7.47 Memory Heaps	430
7.47.1 Detailed Description	430
7.47.2 Macro Definition Documentation	431
7.47.2.1 CH_HEAP_ALIGNMENT	431
7.47.2.2 CH_HEAP_AREA	431
7.47.3 Typedef Documentation	432
7.47.3.1 memory_heap_t	432
7.47.3.2 heap_header_t	432
7.47.4 Function Documentation	432
7.47.4.1 __heap_init()	432
7.47.4.2 chHeapObjectInit()	432
7.47.4.3 chHeapAllocAligned()	433
7.47.4.4 chHeapFree()	433
7.47.4.5 chHeapStatus()	434
7.47.4.6 chHeapAlloc()	434
7.47.4.7 chHeapGetSize()	435
7.47.5 Variable Documentation	436
7.47.5.1 default_heap	436
7.48 Memory Pools	437
7.48.1 Detailed Description	437
7.48.2 Macro Definition Documentation	438
7.48.2.1 __MEMORYPOOL_DATA	439

7.48.2.2 MEMORYPOOL_DECL	439
7.48.2.3 __GUARDEDMEMORYPOOL_DATA	439
7.48.2.4 GUARDEDMEMORYPOOL_DECL	440
7.48.3 Function Documentation	440
7.48.3.1 chPoolObjectInitAligned()	440
7.48.3.2 chPoolLoadArray()	441
7.48.3.3 chPoolAllocI()	442
7.48.3.4 chPoolAlloc()	443
7.48.3.5 chPoolFreeI()	444
7.48.3.6 chPoolFree()	445
7.48.3.7 chGuardedPoolObjectInitAligned()	446
7.48.3.8 chGuardedPoolLoadArray()	447
7.48.3.9 chGuardedPoolAllocTimeoutS()	448
7.48.3.10 chGuardedPoolAllocTimeout()	449
7.48.3.11 chGuardedPoolFree()	450
7.48.3.12 chPoolObjectInit()	451
7.48.3.13 chPoolAdd()	452
7.48.3.14 chPoolAddI()	453
7.48.3.15 chGuardedPoolObjectInit()	454
7.48.3.16 chGuardedPoolGetCounterI()	454
7.48.3.17 chGuardedPoolAllocI()	455
7.48.3.18 chGuardedPoolFreeI()	456
7.48.3.19 chGuardedPoolFreeS()	457
7.48.3.20 chGuardedPoolAdd()	458
7.48.3.21 chGuardedPoolAddI()	459
7.48.3.22 chGuardedPoolAddS()	460
7.49 Complex Services	462
7.49.1 Detailed Description	462
7.50 Objects FIFOs	463
7.50.1 Detailed Description	463
7.50.2 Typedef Documentation	464
7.50.2.1 objects_fifo_t	464
7.50.3 Function Documentation	464
7.50.3.1 chFifoObjectInitAligned()	464
7.50.3.2 chFifoObjectInit()	465
7.50.3.3 chFifoTakeObjectI()	466
7.50.3.4 chFifoTakeObjectTimeoutS()	467
7.50.3.5 chFifoTakeObjectTimeout()	468
7.50.3.6 chFifoReturnObjectI()	469
7.50.3.7 chFifoReturnObjectS()	470
7.50.3.8 chFifoReturnObject()	470
7.50.3.9 chFifoSendObjectI()	471

7.50.3.10 chFifoSendObjectS()	472
7.50.3.11 chFifoSendObject()	473
7.50.3.12 chFifoSendObjectAheadI()	474
7.50.3.13 chFifoSendObjectAheadS()	475
7.50.3.14 chFifoSendObjectAhead()	476
7.50.3.15 chFifoReceiveObjectI()	476
7.50.3.16 chFifoReceiveObjectTimeoutS()	477
7.50.3.17 chFifoReceiveObjectTimeout()	478
7.51 Objects Caches	480
7.51.1 Detailed Description	480
7.51.2 Typedef Documentation	481
7.51.2.1 oc_flags_t	481
7.51.2.2 oc_hash_header_t	481
7.51.2.3 oc_lru_header_t	481
7.51.2.4 oc_object_t	482
7.51.2.5 objects_cache_t	482
7.51.2.6 oc_readf_t	482
7.51.2.7 oc_writef_t	482
7.51.3 Function Documentation	482
7.51.3.1 hash_get_s()	483
7.51.3.2 lru_get_last_s()	483
7.51.3.3 chCacheObjectInit()	484
7.51.3.4 chCacheGetObject()	485
7.51.3.5 chCacheReleaseObjectI()	486
7.51.3.6 chCacheReadObject()	487
7.51.3.7 chCacheWriteObject()	487
7.51.3.8 chCacheReleaseObject()	488
7.52 Dynamic Objects Factory	490
7.52.1 Detailed Description	490
7.52.2 Macro Definition Documentation	493
7.52.2.1 CH_CFG_FACTORY_MAX_NAMES_LENGTH	493
7.52.2.2 CH_CFG_FACTORY_OBJECTS_REGISTRY	493
7.52.2.3 CH_CFG_FACTORY_GENERIC_BUFFERS	493
7.52.2.4 CH_CFG_FACTORY_SEMAPHORES [1/2]	493
7.52.2.5 CH_CFG_FACTORY_SEMAPHORES [2/2]	494
7.52.2.6 CH_CFG_FACTORY_MAILBOXES [1/2]	494
7.52.2.7 CH_CFG_FACTORY_MAILBOXES [2/2]	494
7.52.2.8 CH_CFG_FACTORY_OBJ_FIFOS [1/3]	494
7.52.2.9 CH_CFG_FACTORY_OBJ_FIFOS [2/3]	494
7.52.2.10 CH_CFG_FACTORY_OBJ_FIFOS [3/3]	494
7.52.2.11 CH_CFG_FACTORY_PIPES [1/2]	495
7.52.2.12 CH_CFG_FACTORY_PIPES [2/2]	495

7.52.3 Typedef Documentation	495
7.52.3.1 <code>dyn_element_t</code>	495
7.52.3.2 <code>dyn_list_t</code>	495
7.52.3.3 <code>registered_object_t</code>	495
7.52.3.4 <code>dyn_buffer_t</code>	495
7.52.3.5 <code>dyn_semaphore_t</code>	496
7.52.3.6 <code>dyn_mailbox_t</code>	496
7.52.3.7 <code>dyn_objects_fifo_t</code>	496
7.52.3.8 <code>dyn_pipe_t</code>	496
7.52.3.9 <code>objects_factory_t</code>	496
7.52.4 Function Documentation	496
7.52.4.1 <code>__factory_init()</code>	497
7.52.4.2 <code>chFactoryRegisterObject()</code>	497
7.52.4.3 <code>chFactoryFindObject()</code>	498
7.52.4.4 <code>chFactoryFindObjectByPointer()</code>	498
7.52.4.5 <code>chFactoryReleaseObject()</code>	499
7.52.4.6 <code>chFactoryCreateBuffer()</code>	499
7.52.4.7 <code>chFactoryFindBuffer()</code>	500
7.52.4.8 <code>chFactoryReleaseBuffer()</code>	501
7.52.4.9 <code>chFactoryCreateSemaphore()</code>	501
7.52.4.10 <code>chFactoryFindSemaphore()</code>	502
7.52.4.11 <code>chFactoryReleaseSemaphore()</code>	502
7.52.4.12 <code>chFactoryCreateMailbox()</code>	503
7.52.4.13 <code>chFactoryFindMailbox()</code>	503
7.52.4.14 <code>chFactoryReleaseMailbox()</code>	504
7.52.4.15 <code>chFactoryCreateObjectsFIFO()</code>	504
7.52.4.16 <code>chFactoryFindObjectFIFO()</code>	505
7.52.4.17 <code>chFactoryReleaseObjectsFIFO()</code>	506
7.52.4.18 <code>chFactoryCreatePipe()</code>	506
7.52.4.19 <code>chFactoryFindObjectPipe()</code>	507
7.52.4.20 <code>chFactoryReleasePipe()</code>	507
7.52.4.21 <code>chFactoryDuplicateReference()</code>	509
7.52.4.22 <code>chFactoryGetObject()</code>	509
7.52.4.23 <code>chFactoryGetBufferSize()</code>	510
7.52.4.24 <code>chFactoryGetBuffer()</code>	510
7.52.4.25 <code>chFactoryGetSemaphore()</code>	511
7.52.4.26 <code>chFactoryGetMailbox()</code>	511
7.52.4.27 <code>chFactoryGetObjectsFIFO()</code>	512
7.52.4.28 <code>chFactoryGetPipe()</code>	512
7.52.5 Variable Documentation	513
7.52.5.1 <code>ch_factory</code>	513

8 Data Structure Documentation	515
8.1 ch_binary_semaphore Struct Reference	515
8.1.1 Detailed Description	516
8.2 ch_delta_list Struct Reference	516
8.2.1 Detailed Description	517
8.2.2 Field Documentation	517
8.2.2.1 next	517
8.2.2.2 prev	517
8.2.2.3 delta	517
8.3 ch_dyn_element Struct Reference	518
8.3.1 Detailed Description	518
8.3.2 Field Documentation	518
8.3.2.1 next	518
8.3.2.2 refs	518
8.4 ch_dyn_list Struct Reference	519
8.4.1 Detailed Description	519
8.5 ch_dyn_mailbox Struct Reference	519
8.5.1 Detailed Description	520
8.5.2 Field Documentation	520
8.5.2.1 element	521
8.5.2.2 mbx	521
8.6 ch_dyn_object Struct Reference	521
8.6.1 Detailed Description	522
8.6.2 Field Documentation	522
8.6.2.1 element	522
8.7 ch_dyn_objects_fifo Struct Reference	522
8.7.1 Detailed Description	523
8.7.2 Field Documentation	523
8.7.2.1 element	523
8.7.2.2 fifo	523
8.8 ch_dyn_pipe Struct Reference	523
8.8.1 Detailed Description	525
8.8.2 Field Documentation	525
8.8.2.1 element	525
8.8.2.2 pipe	525
8.9 ch_dyn_semaphore Struct Reference	525
8.9.1 Detailed Description	526
8.9.2 Field Documentation	526
8.9.2.1 element	526
8.9.2.2 sem	526
8.10 ch_job_descriptor Struct Reference	526
8.10.1 Detailed Description	527

8.10.2 Field Documentation	527
8.10.2.1 jobfunc	527
8.10.2.2 jobarg	527
8.11 ch_jobs_queue Struct Reference	527
8.11.1 Detailed Description	528
8.11.2 Field Documentation	528
8.11.2.1 free	529
8.11.2.2 mbx	529
8.12 ch_list Struct Reference	529
8.12.1 Detailed Description	529
8.12.2 Field Documentation	529
8.12.2.1 next	530
8.13 ch_mutex Struct Reference	530
8.13.1 Detailed Description	531
8.13.2 Field Documentation	531
8.13.2.1 queue	531
8.13.2.2 owner	531
8.13.2.3 next	531
8.13.2.4 cnt	531
8.14 ch_objects_cache Struct Reference	532
8.14.1 Detailed Description	533
8.14.2 Field Documentation	533
8.14.2.1 hashn	533
8.14.2.2 hashp	533
8.14.2.3 objn	534
8.14.2.4 objsz	534
8.14.2.5 objvp	534
8.14.2.6 lru	534
8.14.2.7 cache_sem	534
8.14.2.8 lru_sem	534
8.14.2.9 readf	535
8.14.2.10 writef	535
8.15 ch_objects_factory Struct Reference	535
8.15.1 Detailed Description	536
8.15.2 Field Documentation	536
8.15.2.1 mtx	536
8.15.2.2 obj_list	536
8.15.2.3 obj_pool	537
8.15.2.4 buf_list	537
8.15.2.5 sem_list	537
8.15.2.6 sem_pool	537
8.15.2.7 mbx_list	537

8.15.2.8 fifo_list	537
8.15.2.9 pipe_list	538
8.16 ch_objects_fifo Struct Reference	538
8.16.1 Detailed Description	539
8.16.2 Field Documentation	539
8.16.2.1 free	539
8.16.2.2 mbx	539
8.17 ch_oc_hash_header Struct Reference	539
8.17.1 Detailed Description	540
8.17.2 Field Documentation	540
8.17.2.1 hash_next	541
8.17.2.2 hash_prev	541
8.18 ch_oc_lru_header Struct Reference	541
8.18.1 Detailed Description	542
8.18.2 Field Documentation	542
8.18.2.1 hash_next	542
8.18.2.2 hash_prev	542
8.18.2.3 lru_next	542
8.18.2.4 lru_prev	542
8.19 ch_oc_object Struct Reference	543
8.19.1 Detailed Description	544
8.19.2 Field Documentation	544
8.19.2.1 hash_next	544
8.19.2.2 hash_prev	544
8.19.2.3 lru_next	544
8.19.2.4 lru_prev	544
8.19.2.5 obj_group	545
8.19.2.6 obj_key	545
8.19.2.7 obj_sem	545
8.19.2.8 obj_flags	545
8.19.2.9 dptr	545
8.20 ch_os_instance Struct Reference	546
8.20.1 Detailed Description	547
8.20.2 Field Documentation	547
8.20.2.1 rlist	547
8.20.2.2 vlist	547
8.20.2.3 reglist	547
8.20.2.4 core_id	547
8.20.2.5 rfcu	548
8.20.2.6 config	548
8.20.2.7 mainthread	548
8.20.2.8 dbg	548

8.20.2.9 trace_buffer	548
8.20.2.10 kernel_stats	548
8.21 ch_os_instance_config Struct Reference	549
8.21.1 Detailed Description	549
8.21.2 Field Documentation	549
8.21.2.1 name	549
8.21.2.2 mainthread_base	550
8.21.2.3 mainthread_end	550
8.21.2.4 idlethread_base	550
8.21.2.5 idlethread_end	550
8.22 ch_priority_queue Struct Reference	550
8.22.1 Detailed Description	551
8.22.2 Field Documentation	551
8.22.2.1 next	551
8.22.2.2 prev	551
8.22.2.3 prio	551
8.23 ch_queue Struct Reference	552
8.23.1 Detailed Description	552
8.23.2 Field Documentation	552
8.23.2.1 next	552
8.23.2.2 prev	553
8.24 ch_ready_list Struct Reference	553
8.24.1 Detailed Description	554
8.24.2 Field Documentation	554
8.24.2.1 pqueue	554
8.24.2.2 current	554
8.25 ch_registered_static_object Struct Reference	554
8.25.1 Detailed Description	555
8.25.2 Field Documentation	555
8.25.2.1 element	555
8.25.2.2 objp	555
8.26 ch_registry Struct Reference	556
8.26.1 Detailed Description	556
8.26.2 Field Documentation	556
8.26.2.1 queue	556
8.27 ch_rfcu Struct Reference	557
8.27.1 Detailed Description	557
8.27.2 Field Documentation	557
8.27.2.1 mask	557
8.28 ch_semaphore Struct Reference	558
8.28.1 Detailed Description	559
8.28.2 Field Documentation	559

8.28.2.1 queue	559
8.28.2.2 cnt	559
8.29 ch_system Struct Reference	559
8.29.1 Detailed Description	560
8.29.2 Field Documentation	560
8.29.2.1 state	561
8.29.2.2 instances	561
8.29.2.3 tmc	561
8.29.2.4 reglist	561
8.29.2.5 rfcu	561
8.30 ch_system_debug Struct Reference	562
8.30.1 Detailed Description	562
8.30.2 Field Documentation	562
8.30.2.1 panic_msg	562
8.30.2.2 isr_cnt	563
8.30.2.3 lock_cnt	563
8.31 ch_thread Struct Reference	563
8.31.1 Detailed Description	565
8.31.2 Field Documentation	565
8.31.2.1 list	565
8.31.2.2 queue	565
8.31.2.3 pqueue	565
8.31.2.4 hdr	566
8.31.2.5 ctx	566
8.31.2.6 rqueue	566
8.31.2.7 owner	566
8.31.2.8 name	566
8.31.2.9 wabase	566
8.31.2.10 state	567
8.31.2.11 flags	567
8.31.2.12 refs	567
8.31.2.13 ticks	567
8.31.2.14 time	567
8.31.2.15 rdymsg	568
8.31.2.16 exitcode	568
8.31.2.17 wtobjp	568
8.31.2.18 wttrp	568
8.31.2.19 sentmsg	569
8.31.2.20 wtsemp	569
8.31.2.21 wtmtxp	569
8.31.2.22 ewmask	569
8.31.2.23 u	570

8.31.2.24 waiting	570
8.31.2.25 msgqueue	570
8.31.2.26 epending	570
8.31.2.27 mtxlist	570
8.31.2.28 realprio	571
8.31.2.29 mpool	571
8.31.2.30 stats	571
8.32 ch_threads_queue Struct Reference	571
8.32.1 Detailed Description	572
8.32.2 Field Documentation	572
8.32.2.1 queue	572
8.33 ch_virtual_timer Struct Reference	572
8.33.1 Detailed Description	573
8.33.2 Field Documentation	573
8.33.2.1 dlist	573
8.33.2.2 func	573
8.33.2.3 par	573
8.33.2.4 reload	573
8.34 ch_virtual_timers_list Struct Reference	574
8.34.1 Detailed Description	574
8.34.2 Field Documentation	575
8.34.2.1 dlist	575
8.34.2.2 systime	575
8.34.2.3 lasttime	575
8.34.2.4 laststamp	575
8.35 chdebug_t Struct Reference	576
8.35.1 Detailed Description	577
8.35.2 Field Documentation	577
8.35.2.1 identifier	577
8.35.2.2 zero	577
8.35.2.3 size	578
8.35.2.4 version	578
8.35.2.5 ptrsize	578
8.35.2.6 timesize	578
8.35.2.7 threadsize	578
8.35.2.8 off_prio	578
8.35.2.9 off_ctx	579
8.35.2.10 off_newer	579
8.35.2.11 off_older	579
8.35.2.12 off_name	579
8.35.2.13 off_stklimit	579
8.35.2.14 off_state	579

8.35.2.15 off_flags	580
8.35.2.16 off_refs	580
8.35.2.17 off_preempt	580
8.35.2.18 off_time	580
8.36 condition_variable Struct Reference	580
8.36.1 Detailed Description	581
8.36.2 Field Documentation	581
8.36.2.1 queue	581
8.37 event_listener Struct Reference	581
8.37.1 Detailed Description	582
8.37.2 Field Documentation	582
8.37.2.1 next	582
8.37.2.2 listener	582
8.37.2.3 events	582
8.37.2.4 flags	583
8.37.2.5 wflags	583
8.38 event_source Struct Reference	583
8.38.1 Detailed Description	584
8.38.2 Field Documentation	584
8.38.2.1 next	585
8.39 guarded_memory_pool_t Struct Reference	585
8.39.1 Detailed Description	586
8.39.2 Field Documentation	586
8.39.2.1 sem	586
8.39.2.2 pool	586
8.40 heap_header Union Reference	586
8.40.1 Detailed Description	587
8.40.2 Field Documentation	587
8.40.2.1 next	587
8.40.2.2 pages	587
8.40.2.3 heap	587
8.40.2.4 size	587
8.41 kernel_stats_t Struct Reference	588
8.41.1 Detailed Description	588
8.41.2 Field Documentation	589
8.41.2.1 n_irq	589
8.41.2.2 n_ctxswc	589
8.41.2.3 m_crit_thd	589
8.41.2.4 m_crit_isr	589
8.42 mailbox_t Struct Reference	590
8.42.1 Detailed Description	591
8.42.2 Field Documentation	591

8.42.2.1 buffer	591
8.42.2.2 top	591
8.42.2.3 wrptr	591
8.42.2.4 rdptr	592
8.42.2.5 cnt	592
8.42.2.6 reset	592
8.42.2.7 qw	592
8.42.2.8 qr	592
8.43 memcore_t Struct Reference	593
8.43.1 Detailed Description	593
8.43.2 Field Documentation	593
8.43.2.1 basemem	593
8.43.2.2 topmem	593
8.44 memory_heap Struct Reference	594
8.44.1 Detailed Description	595
8.44.2 Field Documentation	595
8.44.2.1 provider	595
8.44.2.2 header	595
8.44.2.3 mtx	595
8.45 memory_pool_t Struct Reference	596
8.45.1 Detailed Description	596
8.45.2 Field Documentation	596
8.45.2.1 next	597
8.45.2.2 object_size	597
8.45.2.3 align	597
8.45.2.4 provider	597
8.46 pipe_t Struct Reference	597
8.46.1 Detailed Description	599
8.46.2 Field Documentation	599
8.46.2.1 buffer	599
8.46.2.2 top	599
8.46.2.3 wrptr	600
8.46.2.4 rdptr	600
8.46.2.5 cnt	600
8.46.2.6 reset	600
8.46.2.7 wtr	600
8.46.2.8 rtr	600
8.46.2.9 cmtx	601
8.46.2.10 wmtx	601
8.46.2.11 rmtx	601
8.47 pool_header Struct Reference	601
8.47.1 Detailed Description	602

8.47.2 Field Documentation	602
8.47.2.1 next	602
8.48 thread_descriptor_t Struct Reference	602
8.48.1 Detailed Description	603
8.48.2 Field Documentation	603
8.48.2.1 name	603
8.48.2.2 wbase	603
8.48.2.3 wend	603
8.48.2.4 prio	604
8.48.2.5 funcp	604
8.48.2.6 arg	604
8.48.2.7 instance	604
8.49 time_measurement_t Struct Reference	604
8.49.1 Detailed Description	605
8.49.2 Field Documentation	605
8.49.2.1 best	605
8.49.2.2 worst	605
8.49.2.3 last	606
8.49.2.4 n	606
8.49.2.5 cumulative	606
8.50 tm_calibration_t Struct Reference	606
8.50.1 Detailed Description	607
8.50.2 Field Documentation	607
8.50.2.1 offset	607
8.51 trace_buffer_t Struct Reference	607
8.51.1 Detailed Description	609
8.51.2 Field Documentation	609
8.51.2.1 suspended	609
8.51.2.2 size	609
8.51.2.3 ptr	609
8.51.2.4 buffer	609
8.52 trace_event_t Struct Reference	610
8.52.1 Detailed Description	611
8.52.2 Field Documentation	612
8.52.2.1 type	612
8.52.2.2 state	612
8.52.2.3 rtstamp	612
8.52.2.4 time	612
8.52.2.5 ntp	612
8.52.2.6 wtobjp	613
8.52.2.7 sw	613
8.52.2.8 tp	613

8.52.2.9 msg	613
8.52.2.10 rdy	613
8.52.2.11 name	613
8.52.2.12 isr	614
8.52.2.13 reason	614
8.52.2.14 halt	614
8.52.2.15 up1	614
8.52.2.16 up2	614
8.52.2.17 user	614
9 File Documentation	615
9.1 ch.h File Reference	615
9.1.1 Detailed Description	616
9.2 chalign.h File Reference	616
9.2.1 Detailed Description	616
9.3 chbsem.h File Reference	617
9.3.1 Detailed Description	618
9.4 chchecks.h File Reference	618
9.4.1 Detailed Description	618
9.5 chcond.c File Reference	618
9.5.1 Detailed Description	619
9.6 chcond.h File Reference	619
9.6.1 Detailed Description	620
9.7 chconf.h File Reference	620
9.7.1 Detailed Description	623
9.8 chccustomer.h File Reference	623
9.8.1 Detailed Description	624
9.9 chdebug.c File Reference	624
9.9.1 Detailed Description	625
9.10 chdebug.h File Reference	625
9.10.1 Detailed Description	626
9.11 chdelegates.c File Reference	626
9.11.1 Detailed Description	626
9.12 chdelegates.h File Reference	627
9.12.1 Detailed Description	628
9.13 chdynamic.c File Reference	628
9.13.1 Detailed Description	628
9.14 chdynamic.h File Reference	628
9.14.1 Detailed Description	628
9.15 clearly.h File Reference	629
9.15.1 Detailed Description	630
9.16 chevents.c File Reference	630

9.16.1 Detailed Description	631
9.17 chevents.h File Reference	631
9.17.1 Detailed Description	632
9.18 chfactory.c File Reference	633
9.18.1 Detailed Description	634
9.19 chfactory.h File Reference	634
9.19.1 Detailed Description	636
9.20 chinstances.c File Reference	637
9.20.1 Detailed Description	637
9.21 chinstances.h File Reference	637
9.21.1 Detailed Description	637
9.22 chjobs.h File Reference	637
9.22.1 Detailed Description	639
9.23 chlib.h File Reference	639
9.23.1 Detailed Description	640
9.24 chlicense.h File Reference	640
9.24.1 Detailed Description	641
9.25 chlists.h File Reference	641
9.25.1 Detailed Description	643
9.26 chmboxes.c File Reference	643
9.26.1 Detailed Description	643
9.27 chmboxes.h File Reference	643
9.27.1 Detailed Description	645
9.28 chmemcore.c File Reference	645
9.28.1 Detailed Description	645
9.29 chmemcore.h File Reference	645
9.29.1 Detailed Description	646
9.30 chmemheaps.c File Reference	646
9.30.1 Detailed Description	647
9.31 chmemheaps.h File Reference	647
9.31.1 Detailed Description	648
9.32 chmpools.c File Reference	648
9.32.1 Detailed Description	649
9.33 chmpools.h File Reference	649
9.33.1 Detailed Description	650
9.34 chmsg.c File Reference	650
9.34.1 Detailed Description	651
9.35 chmsg.h File Reference	651
9.35.1 Detailed Description	651
9.36 chmtx.c File Reference	651
9.36.1 Detailed Description	652
9.37 chmtx.h File Reference	652

9.37.1 Detailed Description	653
9.38 chobjcaches.c File Reference	653
9.38.1 Detailed Description	654
9.39 chobjcaches.h File Reference	654
9.39.1 Detailed Description	656
9.40 chobjects.h File Reference	656
9.40.1 Detailed Description	657
9.41 chobjfifos.h File Reference	657
9.41.1 Detailed Description	658
9.42 chpipes.c File Reference	658
9.42.1 Detailed Description	659
9.43 chpipes.h File Reference	659
9.43.1 Detailed Description	660
9.44 chport.h File Reference	660
9.44.1 Detailed Description	660
9.45 chregistry.c File Reference	660
9.45.1 Detailed Description	661
9.46 chregistry.h File Reference	661
9.46.1 Detailed Description	662
9.47 chrestrictions.h File Reference	662
9.47.1 Detailed Description	662
9.48 chrfcu.c File Reference	662
9.48.1 Detailed Description	663
9.49 chrfcu.h File Reference	663
9.49.1 Detailed Description	663
9.50 chschd.c File Reference	664
9.50.1 Detailed Description	664
9.51 chschd.h File Reference	664
9.51.1 Detailed Description	667
9.52 chsem.c File Reference	667
9.52.1 Detailed Description	668
9.53 chsem.h File Reference	668
9.53.1 Detailed Description	669
9.54 chstats.c File Reference	669
9.54.1 Detailed Description	669
9.55 chstats.h File Reference	670
9.55.1 Detailed Description	670
9.56 chsys.c File Reference	670
9.56.1 Detailed Description	671
9.57 chsys.h File Reference	671
9.57.1 Detailed Description	673
9.58 chthreads.c File Reference	673

9.58.1 Detailed Description	675
9.59 chthreads.h File Reference	675
9.59.1 Detailed Description	677
9.60 chtime.h File Reference	678
9.60.1 Detailed Description	679
9.61 chtm.c File Reference	679
9.61.1 Detailed Description	680
9.62 chtm.h File Reference	680
9.62.1 Detailed Description	680
9.63 chtrace.c File Reference	680
9.63.1 Detailed Description	681
9.64 chtrace.h File Reference	681
9.64.1 Detailed Description	683
9.65 chversion.h File Reference	683
9.65.1 Detailed Description	683
9.66 chvt.c File Reference	683
9.66.1 Detailed Description	684
9.67 chvt.h File Reference	684
9.67.1 Detailed Description	685
Index	687

Chapter 1

ChibiOS/RT

1.1 Copyright

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

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

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

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

1.2 Introduction

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

1.3 Related Documents

- ChibiOS/RT General Architecture

Chapter 2

Kernel Concepts

ChibiOS Kernel Concepts This article applies to both RT and NIL kernels, many concepts are also applicable to HAL (states and API classes).

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

2.1 Naming Conventions

ChibiOS/RT and ChibiOS/NIL APIs are named following this convention: `ch<group><action><suffix>()`. Examples of groups are: `Sys`, `Sch`, `Time`, `VT`, `Thd`, `Sem`, etc.

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](#). This kind of functions never reschedule internally and are meant to be used from interrupt handlers. If used from thread level then an explicit reschedule operation could be necessary before leaving the containing critical zone.
- "**S**", S-Class APIs are invokable only from the **S-Locked** state. See [System States](#). This kind of functions can reschedule internally but are not guaranteed to do so necessarily. For example a wait function on a semaphore can reschedule or not reschedule depending on the semaphore state.
- "**X**", X-Class APIs are invokable from any context.

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

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.

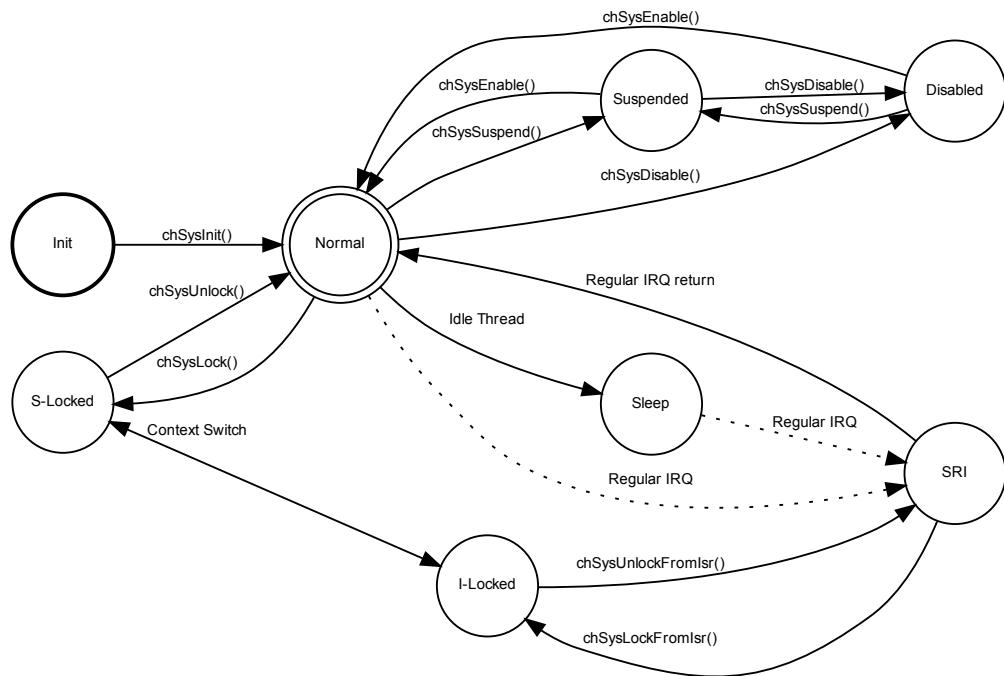
The mapping of the above logical classes into physical interrupts priorities is, of course, port dependent. See the documentation of the various ports for details.

2.4 System States

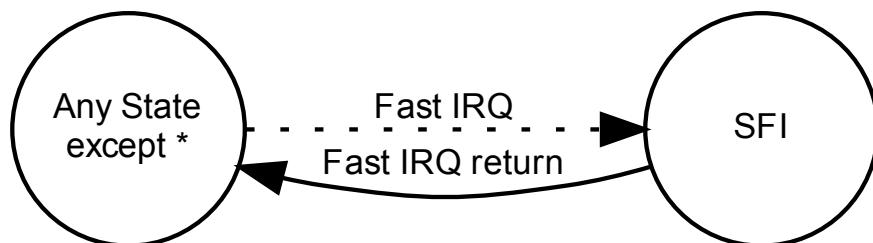
When using ChibiOS/RT the system can be in one of the following logical operating states:

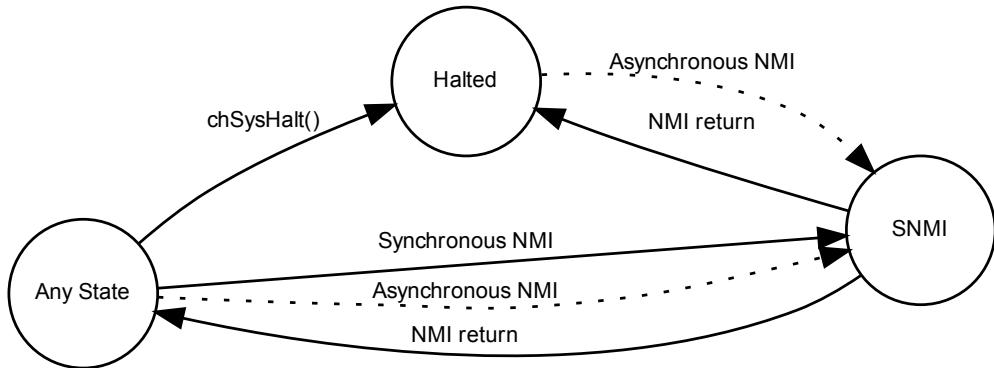
- **Init.** When the system is in this state all the maskable interrupt sources are disabled. In this state it is not possible to use any system API except `chSysInit()`. This state is entered after a physical reset.
- **Normal.** All the interrupt sources are enabled and the system APIs are accessible,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

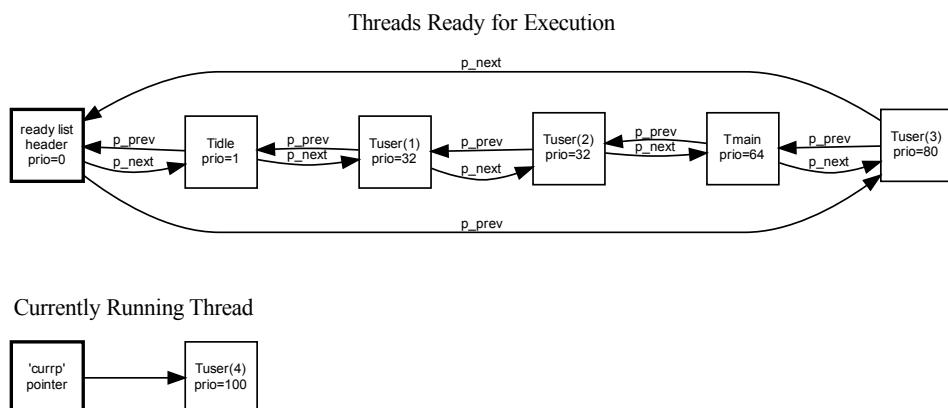


**Attention**

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

2.5 Scheduling

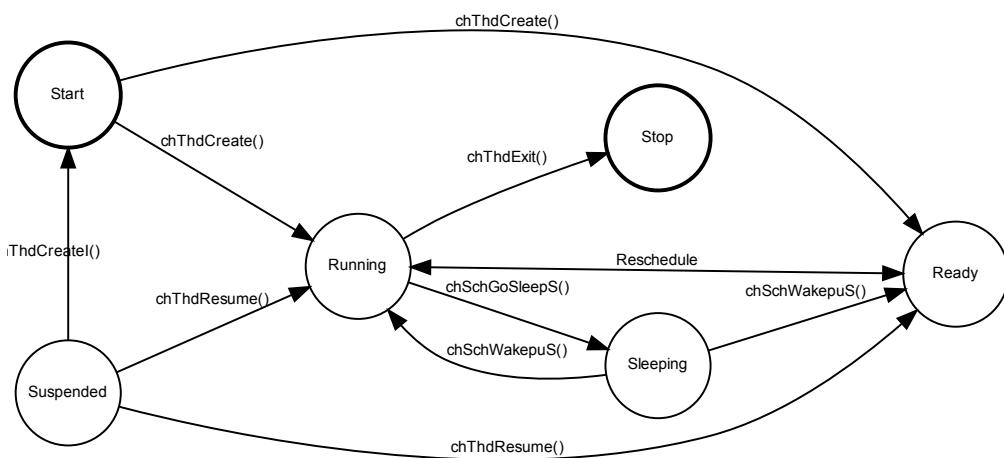
The strategy is very simple the currently ready thread with the highest priority is executed. If more than one thread with equal priority are eligible for execution then they are executed in a round-robin way, the CPU time slice constant is configurable. The ready list is a double linked list of threads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



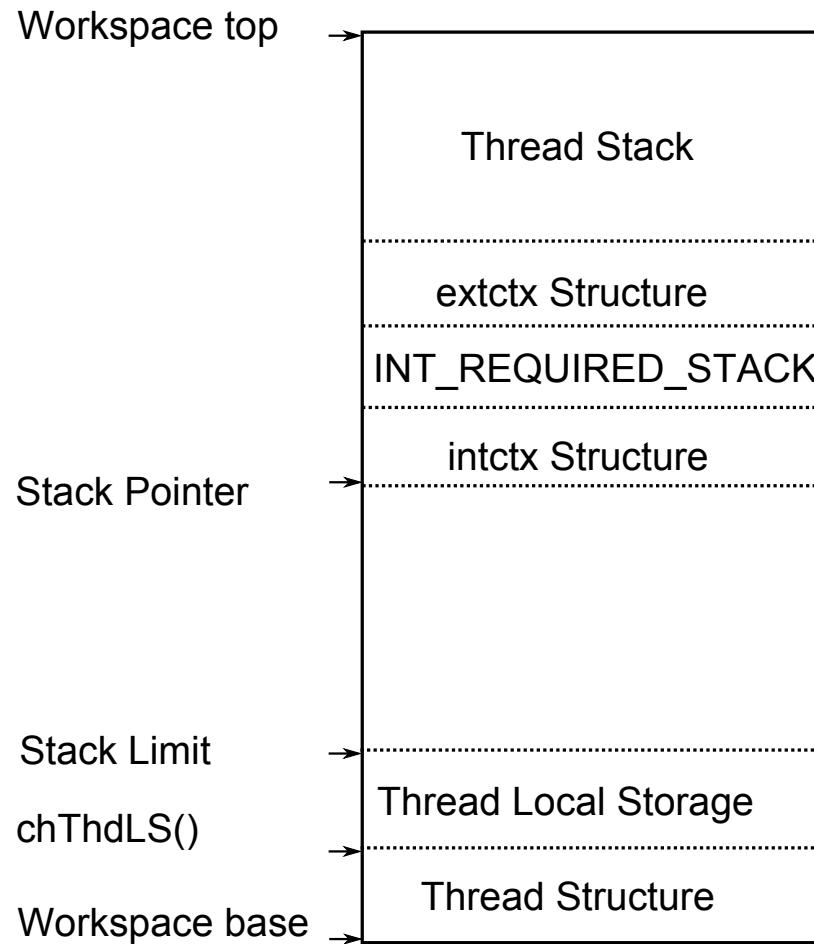
2.7 Priority Levels

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.
- HIGHPRIORIO, the highest priority level that can be assigned to an user thread.
- ABSPRIO, absolute maximum software priority level, it can be higher than HIGHPRIORIO but the numerical values above HIGHPRIORIO 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. 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

Module Index

3.1 Modules

Here is a list of all modules:

Release and Licensing	21
Release Information	22
Customer Information	24
License Settings	26
RT Kernel	28
Version Numbers and Identification	29
Configuration	31
Options	32
Checks	54
Restrictions	55
System	56
Port Interface	57
OS Types and Structures	58
OS Instances	68
Runtime Faults Collection Unit	71
Lists and Queues	74
Scheduler	89
Base Kernel Services	110
System Management	111
Memory Alignment	136
Time and Intervals	139
Virtual Timers	153
Threads	180
Time Measurement	223
Synchronization	227
Counting Semaphores	228
Mutexes	246
Condition Variables	261
Event Flags	273
Synchronous Messages	300
Dynamic Threads	311
Registry	314
Debug	324
Checks and Assertions	325

Tracing	335
Statistics	345
OS Library	349
Version Numbers and Identification	350
Synchronization	353
Binary Semaphores	354
Mailboxes	365
Pipes	383
Delegate Threads	391
Jobs Queues	403
Memory Management	418
Core Memory Manager	419
Memory Heaps	430
Memory Pools	437
Complex Services	462
Objects FIFOs	463
Objects Caches	480
Dynamic Objects Factory	490

Chapter 4

Hierarchical Index

4.1 Class Hierarchy

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

ch_delta_list	516
ch_dyn_element	518
ch_dyn_list	519
ch_dyn_mailbox	519
ch_dyn_object	521
ch_dyn_objects_fifo	522
ch_dyn_pipe	523
ch_dyn_semaphore	525
ch_job_descriptor	526
ch_jobs_queue	527
ch_list	529
ch_mutex	530
ch_objects_cache	532
ch_objects_factory	535
ch_objects_fifo	538
ch_oc_hash_header	539
ch_oc_iru_header	541
ch_oc_object	543
ch_os_instance	546
ch_os_instance_config	549
ch_priority_queue	550
ch_queue	552
ch_ready_list	553
ch_registered_static_object	554
ch_registry	556
ch_rfcu	557
ch_semaphore	558
ch_binary_semaphore	515
ch_system	559
ch_system_debug	562
ch_thread	563
ch_threads_queue	571
ch_virtual_timer	572
ch_virtual_timers_list	574
chdebug_t	576

condition_variable	580
event_listener	581
event_source	583
guarded_memory_pool_t	585
heap_header	586
kernel_stats_t	588
mailbox_t	590
memcore_t	593
memory_heap	594
memory_pool_t	596
pipe_t	597
pool_header	601
thread_descriptor_t	602
time_measurement_t	604
tm_calibration_t	606
trace_buffer_t	607
trace_event_t	610

Chapter 5

Data Structure Index

5.1 Data Structures

Here are the data structures with brief descriptions:

ch_binary_semaphore	Binary semaphore type	515
ch_delta_list	Delta list element and header structure	516
ch_dyn_element	Type of a dynamic object list element	518
ch_dyn_list	Type of a dynamic object list	519
ch_dyn_mailbox	Type of a dynamic buffer object	519
ch_dyn_object	Type of a dynamic buffer object	521
ch_dyn_objects_fifo	Type of a dynamic buffer object	522
ch_dyn_pipe	Type of a dynamic pipe object	523
ch_dyn_semaphore	Type of a dynamic semaphore	525
ch_job_descriptor	Type of a job descriptor	526
ch_jobs_queue	Type of a jobs queue	527
ch_list	Structure representing a generic single link list header and element	529
ch_mutex	Mutex structure	530
ch_objects_cache	Structure representing a cache object	532
ch_objects_factory	Type of the factory main object	535
ch_objects_fifo	Type of an objects FIFO	538
ch_oc_hash_header	Structure representing an hash table element	539
ch_oc_lru_header	Structure representing an hash table element	541

<code>ch_oc_object</code>	Structure representing a cached object	543
<code>ch_os_instance</code>	System instance data structure	546
<code>ch_os_instance_config</code>	Type of an system instance configuration	549
<code>ch_priority_queue</code>	Structure representing a generic priority-ordered bidirectional linked list header and element	550
<code>ch_queue</code>	Structure representing a generic bidirectional linked list header and element	552
<code>ch_ready_list</code>	Type of a ready list header	553
<code>ch_registered_static_object</code>	Type of a registered object	554
<code>ch_registry</code>	Type of a registry structure	556
<code>ch_rfcu</code>	Type of an RFCU structure	557
<code>ch_semaphore</code>	Semaphore structure	558
<code>ch_system</code>	Type of system data structure	559
<code>ch_system_debug</code>	System debug data structure	562
<code>ch_thread</code>	Structure representing a thread	563
<code>ch_threads_queue</code>	Type of a threads queue	571
<code>ch_virtual_timer</code>	Structure representing a Virtual Timer	572
<code>ch_virtual_timers_list</code>	Type of virtual timers list header	574
<code>chdebug_t</code>	ChibiOS/RT memory signature record	576
<code>condition_variable</code>	Condition_variable_t structure	580
<code>event_listener</code>	Event Listener structure	581
<code>event_source</code>	Event Source structure	583
<code>guarded_memory_pool_t</code>	Guarded memory pool descriptor	585
<code>heap_header</code>	Memory heap block header	586
<code>kernel_stats_t</code>	Type of a kernel statistics structure	588
<code>mailbox_t</code>	Structure representing a mailbox object	590
<code>memcore_t</code>	Type of memory core object	593
<code>memory_heap</code>	Structure describing a memory heap	594
<code>memory_pool_t</code>	Memory pool descriptor	596
<code>pipe_t</code>	Structure representing a pipe object	597
<code>pool_header</code>	Memory pool free object header	601

thread_descriptor_t	Type of a thread descriptor	602
time_measurement_t	Type of a Time Measurement object	604
tm_calibration_t	Type of a time measurement calibration data	606
trace_buffer_t	Trace buffer header	607
trace_event_t	Trace buffer record	610

Chapter 6

File Index

6.1 File List

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

ch.h	ChibiOS/RT main include file	615
chalign.h	Memory alignment macros and structures	616
chbsem.h	Binary semaphores structures and macros	617
chchecks.h	Configuration file checks header	618
chcond.c	Condition Variables code	618
chcond.h	Condition Variables macros and structures	619
chconf.h	Configuration file template	620
chcustomer.h	Customer-related info	623
chdebug.c	Debug support code	624
chdebug.h	Debug support macros and structures	625
chdelegates.c	Delegate threads code	626
chdelegates.h	Delegate threads macros and structures	627
chdynamic.c	Dynamic threads code	628
chdynamic.h	Dynamic threads macros and structures	628
clearly.h	Early forward types declarations header	629
chevents.c	Events code	630
chevents.h	Events macros and structures	631
chfactory.c	ChibiOS objects factory and registry code	633

chfactory.h	
ChibiOS objects factory structures and macros	634
chinstances.c	
OS instances code	637
chinstances.h	
OS instances macros and structures	637
chjobs.h	
Jobs Queues structures and macros	637
chlib.h	
ChibiOS/LIB main include file	639
chlicense.h	
License Module macros and structures	640
chlists.h	
Lists and Queues header	641
chmboxes.c	
Mailboxes code	643
chmboxes.h	
Mailboxes macros and structures	643
chmemcore.c	
Core memory manager code	645
chmemcore.h	
Core memory manager macros and structures	645
chmemheaps.c	
Memory heaps code	646
chmemheaps.h	
Memory heaps macros and structures	647
chmempools.c	
Memory Pools code	648
chmempools.h	
Memory Pools macros and structures	649
chmsg.c	
Messages code	650
chmsg.h	
Messages macros and structures	651
chmtx.c	
Mutexes code	651
chmtx.h	
Mutexes macros and structures	652
chobjcaches.c	
Objects Caches code	653
chobjcaches.h	
Objects Caches macros and structures	654
chobjects.h	
Operating System Objects macros and structures	656
chobjfifos.h	
Objects FIFO structures and macros	657
chpipes.c	
Pipes code	658
chpipes.h	
Pipes macros and structures	659
chport.h	
Port wrapper header	660
chregistry.c	
Threads registry code	660
chregistry.h	
Threads registry macros and structures	661
chrestrictions.h	
Licensing restrictions header	662

chrfcu.c	Runtime Faults Collection Unit code	662
chrfcu.h	Runtime Faults Collection Unit macros and structures	663
chsched.c	Scheduler code	664
chsched.h	Scheduler macros and structures	664
chsem.c	Semaphores code	667
chsem.h	Semaphores macros and structures	668
chstats.c	Statistics module code	669
chstats.h	Statistics module macros and structures	670
chsys.c	System related code	670
chsys.h	System related macros and structures	671
cthreads.c	Threads code	673
cthreads.h	Threads module macros and structures	675
ctime.h	Time and intervals macros and structures	678
ctm.c	Time Measurement module code	679
ctm.h	Time Measurement module macros and structures	680
ctrace.c	Tracer code	680
ctrace.h	Tracer macros and structures	681
chversion.h	Version Module macros and structures	683
chvt.c	Time and Virtual Timers module code	683
chvt.h	Time and Virtual Timers module macros and structures	684

Chapter 7

Module Documentation

7.1 Release and Licensing

7.1.1 Detailed Description

Modules

- Release Information
- Customer Information
- License Settings

7.2 Release Information

7.2.1 Detailed Description

This module contains information about the ChibiOS release, it is common to all subsystems.

Macros

- `#define __CHIBIOS__`
ChibiOS product identification macro.
- `#define CH_VERSION_STABLE 1`
Stable release flag.
- `#define CH_VERSION_DATE (((CH_VERSION_YEAR + 2000) * 100) + CH_VERSION_MONTH)`
Current version date in numeric form (yyyymm).

ChibiOS version identification

- `#define CH_VERSION "21.6.0"`
ChibiOS version string.
- `#define CH_VERSION_YEAR 21`
ChibiOS version release year.
- `#define CH_VERSION_MONTH 6`
ChibiOS version release month.
- `#define CH_VERSION_PATCH 0`
ChibiOS version patch number.
- `#define CH_VERSION_NICKNAME "Atrani"`
ChibiOS version nickname.

7.2.2 Macro Definition Documentation

7.2.2.1 __CHIBIOS__

```
#define __CHIBIOS__
```

ChibiOS product identification macro.

7.2.2.2 CH_VERSION_STABLE

```
#define CH_VERSION_STABLE 1
```

Stable release flag.

7.2.2.3 CH_VERSION

```
#define CH_VERSION "21.6.0"
```

ChibiOS version string.

7.2.2.4 CH_VERSION_YEAR

```
#define CH_VERSION_YEAR 21
```

ChibiOS version release year.

7.2.2.5 CH_VERSION_MONTH

```
#define CH_VERSION_MONTH 6
```

ChibiOS version release month.

7.2.2.6 CH_VERSION_PATCH

```
#define CH_VERSION_PATCH 0
```

ChibiOS version patch number.

7.2.2.7 CH_VERSION_NICKNAME

```
#define CH_VERSION_NICKNAME "Atrani"
```

ChibiOS version nickname.

7.2.2.8 CH_VERSION_DATE

```
#define CH_VERSION_DATE (((CH_VERSION_YEAR + 2000) * 100) + CH_VERSION_MONTH)
```

Current version date in numeric form (yyyymm).

7.3 Customer Information

7.3.1 Detailed Description

This module encapsulates licensee information, this is only meaningful for commercial licenses. It is a stub for public releases.

Macros

- #define CH_CUSTOMER_ID_STRING "Santa, North Pole"
Customer readable identifier.
- #define CH_CUSTOMER_ID_CODE "xxxx-yyyy"
Customer code.
- #define CH_CUSTOMER_LICENSE_EOS_DATE 209912
End-Of-Support date (yyyymm).
- #define CH_CUSTOMER_LICENSE_VERSION_YEAR 99
Licensed branch year.
- #define CH_CUSTOMER_LICENSE_VERSION_MONTH 12
Licensed branch month.
- #define CH_LICENSE CH_LICENSE_GPL
Current license.
- #define CH_CUSTOMER_LICENSE_VERSION_DATE
Licensed version date in numeric form (yyyymm).

Licensed Products

- #define CH_CUSTOMER_LIC_RT TRUE
- #define CH_CUSTOMER_LIC_NIL TRUE
- #define CH_CUSTOMER_LIC_OSLIB TRUE
- #define CH_CUSTOMER_LIC_EX TRUE
- #define CH_CUSTOMER_LIC_SB TRUE
- #define CH_CUSTOMER_LIC_PORT_CM0 TRUE
- #define CH_CUSTOMER_LIC_PORT_CM3 TRUE
- #define CH_CUSTOMER_LIC_PORT_CM4 TRUE
- #define CH_CUSTOMER_LIC_PORT_CM7 TRUE
- #define CH_CUSTOMER_LIC_PORT_CM33 TRUE
- #define CH_CUSTOMER_LIC_PORT_ARM79 TRUE
- #define CH_CUSTOMER_LIC_PORT_E200Z0 TRUE
- #define CH_CUSTOMER_LIC_PORT_E200Z2 TRUE
- #define CH_CUSTOMER_LIC_PORT_E200Z3 TRUE
- #define CH_CUSTOMER_LIC_PORT_E200Z4 TRUE

7.3.2 Macro Definition Documentation

7.3.2.1 CH_CUSTOMER_ID_STRING

```
#define CH_CUSTOMER_ID_STRING "Santa, North Pole"
```

Customer readable identifier.

7.3.2.2 CH_CUSTOMER_ID_CODE

```
#define CH_CUSTOMER_ID_CODE "xxxx-yyyy"
```

Customer code.

7.3.2.3 CH_CUSTOMER_LICENSE_EOS_DATE

```
#define CH_CUSTOMER_LICENSE_EOS_DATE 209912
```

End-Of-Support date (yyymm).

7.3.2.4 CH_CUSTOMER_LICENSE_VERSION_YEAR

```
#define CH_CUSTOMER_LICENSE_VERSION_YEAR 99
```

Licensed branch year.

7.3.2.5 CH_CUSTOMER_LICENSE_VERSION_MONTH

```
#define CH_CUSTOMER_LICENSE_VERSION_MONTH 12
```

Licensed branch month.

7.3.2.6 CH_LICENSE

```
#define CH_LICENSE CH_LICENSE_GPL
```

Current license.

Note

This setting is reserved to the copyright owner.

Changing this setting invalidates the license.

The license statement in the source headers is valid, applicable and binding regardless this setting.

7.3.2.7 CH_CUSTOMER_LICENSE_VERSION_DATE

```
#define CH_CUSTOMER_LICENSE_VERSION_DATE
```

Value:

```
((CH_CUSTOMER_LICENSE_VERSION_YEAR + 2000) * 100) +  
CH_CUSTOMER_LICENSE_VERSION_MONTH \
```

Licensed version date in numeric form (yyymm).

7.4 License Settings

7.4.1 Detailed Description

This module contains all the definitions required for defining a licensing scheme for customers or public releases.

Macros

- `#define CH_LICENSE_TYPE_STRING "GNU General Public License 3 (GPL3)"`
License identification string.
- `#define CH_LICENSE_ID_STRING "N/A"`
Customer identification string.
- `#define CH_LICENSE_ID_CODE "N/A"`
Customer code.
- `#define CH_LICENSE_MODIFIABLE_CODE TRUE`
Code modifiability restrictions.
- `#define CH_LICENSE_FEATURES CH_FEATURES_FULL`
Code functionality restrictions.
- `#define CH_LICENSE_MAX_DEPLOY CH_DEPLOY_UNLIMITED`
Code deploy restrictions.

Allowed Features Levels

- `#define CH_FEATURES_BASIC 0`
- `#define CH_FEATURES_INTERMEDIATE 1`
- `#define CH_FEATURES_FULL 2`

Deployment Options

- `#define CH_DEPLOY_UNLIMITED -1`
- `#define CH_DEPLOY_NONE 0`

Licensing Options

- `#define CH_LICENSE_GPL 0`
- `#define CH_LICENSE_GPL_EXCEPTION 1`
- `#define CH_LICENSE_COMMERCIAL_FREE 2`
- `#define CH_LICENSE_COMMERCIAL_DEV_1000 3`
- `#define CH_LICENSE_COMMERCIAL_DEV_5000 4`
- `#define CH_LICENSE_COMMERCIAL_FULL 5`
- `#define CH_LICENSE_COMMERCIAL_RUNTIME 6`
- `#define CH_LICENSE_PARTNER 7`

7.4.2 Macro Definition Documentation

7.4.2.1 CH_LICENSE_TYPE_STRING

```
#define CH_LICENSE_TYPE_STRING "GNU General Public License 3 (GPL3)"
```

License identification string.

This string identifies the license in a machine-readable format.

7.4.2.2 CH_LICENSE_ID_STRING

```
#define CH_LICENSE_ID_STRING "N/A"
```

Customer identification string.

This information is only available for registered commercial users.

7.4.2.3 CH_LICENSE_ID_CODE

```
#define CH_LICENSE_ID_CODE "N/A"
```

Customer code.

This information is only available for registered commercial users.

7.4.2.4 CH_LICENSE_MODIFIABLE_CODE

```
#define CH_LICENSE_MODIFIABLE_CODE TRUE
```

Code modifiability restrictions.

This setting defines if the source code is user-modifiable or not.

7.4.2.5 CH_LICENSE_FEATURES

```
#define CH_LICENSE_FEATURES CH_FEATURES_FULL
```

Code functionality restrictions.

7.4.2.6 CH_LICENSE_MAX_DEPLOY

```
#define CH_LICENSE_MAX_DEPLOY CH_DEPLOY_UNLIMITED
```

Code deploy restrictions.

This is the per-core deploy limit allowed under the current license scheme.

7.5 RT Kernel

7.5.1 Detailed Description

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

Modules

- [Version Numbers and Identification](#)
- [Configuration](#)
- [System](#)
- [Base Kernel Services](#)
- [Synchronization](#)
- [Dynamic Threads](#)
- [Registry](#)
- [Debug](#)

7.6 Version Numbers and Identification

7.6.1 Detailed Description

This header includes all the required kernel headers so it is the only kernel header you usually want to include in your application.

Kernel related info.

Macros

- `#define __CHIBIOS_RT__`
ChibiOS/RT identification macro.
- `#define CH_KERNEL_STABLE 0`
Stable release flag.

ChibiOS/RT version identification

- `#define CH_KERNEL_VERSION "7.0.0"`
Kernel version string.
- `#define CH_KERNEL_MAJOR 7`
Kernel version major number.
- `#define CH_KERNEL_MINOR 0`
Kernel version minor number.
- `#define CH_KERNEL_PATCH 0`
Kernel version patch number.

Constants for configuration options

- `#define FALSE 0`
Generic 'false' preprocessor boolean constant.
- `#define TRUE 1`
Generic 'true' preprocessor boolean constant.

7.6.2 Macro Definition Documentation

7.6.2.1 __CHIBIOS_RT__

```
#define __CHIBIOS_RT__
```

ChibiOS/RT identification macro.

7.6.2.2 CH_KERNEL_STABLE

```
#define CH_KERNEL_STABLE 0
```

Stable release flag.

7.6.2.3 CH_KERNEL_VERSION

```
#define CH_KERNEL_VERSION "7.0.0"
```

Kernel version string.

7.6.2.4 CH_KERNEL_MAJOR

```
#define CH_KERNEL_MAJOR 7
```

Kernel version major number.

7.6.2.5 CH_KERNEL_MINOR

```
#define CH_KERNEL_MINOR 0
```

Kernel version minor number.

7.6.2.6 CH_KERNEL_PATCH

```
#define CH_KERNEL_PATCH 0
```

Kernel version patch number.

7.6.2.7 FALSE

```
#define FALSE 0
```

Generic 'false' preprocessor boolean constant.

Note

It is meant to be used in configuration files as switch.

7.6.2.8 TRUE

```
#define TRUE 1
```

Generic 'true' preprocessor boolean constant.

Note

It is meant to be used in configuration files as switch.

7.7 Configuration

7.7.1 Detailed Description

Modules

- Options
- Checks
- Restrictions

7.8 Options

7.8.1 Detailed Description

Kernel related settings and hooks.

System settings

- #define CH_CFG_SMP_MODE FALSE

Handling of instances.

System timers settings

- #define CH_CFG_ST_RESOLUTION 32
System time counter resolution.
- #define CH_CFG_ST_FREQUENCY 10000
System tick frequency.
- #define CH_CFG_INTERVALS_SIZE 32
Time intervals data size.
- #define CH_CFG_TIME_TYPES_SIZE 32
Time types data size.
- #define CH_CFG_ST_TIMDELTA 2
Time delta constant for the tick-less mode.

Kernel parameters and options

- #define CH_CFG_TIME_QUANTUM 0
Round robin interval.
- #define CH_CFG_NO_IDLE_THREAD FALSE
Idle thread automatic spawn suppression.

Performance options

- #define CH_CFG_OPTIMIZE_SPEED TRUE
OS optimization.

Subsystem options

- #define CH_CFG_USE_TM TRUE
Time Measurement APIs.
- #define CH_CFG_USE_TIMESTAMP TRUE
Time Stamps APIs.
- #define CH_CFG_USE_REGISTRY TRUE
Threads registry APIs.
- #define CH_CFG_USE_WAITEXIT TRUE
Threads synchronization APIs.
- #define CH_CFG_USE_SEMAPHORES TRUE
Semaphores APIs.
- #define CH_CFG_USE_SEMAPHORES_PRIORITY FALSE
Semaphores queuing mode.
- #define CH_CFG_USE_MUTEXES TRUE
Mutexes APIs.
- #define CH_CFG_USE_MUTEXES_RECURSIVE FALSE
Enables recursive behavior on mutexes.
- #define CH_CFG_USE_CONDVARS TRUE
Conditional Variables APIs.
- #define CH_CFG_USE_CONDVARS_TIMEOUT TRUE
Conditional Variables APIs with timeout.
- #define CH_CFG_USE_EVENTS TRUE
Events Flags APIs.
- #define CH_CFG_USE_EVENTS_TIMEOUT TRUE
Events Flags APIs with timeout.
- #define CH_CFG_USE_MESSAGES TRUE
Synchronous Messages APIs.
- #define CH_CFG_USE_MESSAGES_PRIORITY FALSE
Synchronous Messages queuing mode.
- #define CH_CFG_USE_DYNAMIC TRUE
Dynamic Threads APIs.

OSLIB options

- #define CH_CFG_USE_MAILBOXES TRUE
Mailboxes APIs.
- #define CH_CFG_USE_MEMCORE TRUE
Core Memory Manager APIs.
- #define CH_CFG_MEMCORE_SIZE 0
Managed RAM size.
- #define CH_CFG_USE_HEAP TRUE
Heap Allocator APIs.
- #define CH_CFG_USE_MEMPOOLS TRUE
Memory Pools Allocator APIs.
- #define CH_CFG_USE_OBJ_FIFOS TRUE
Objects FIFOs APIs.
- #define CH_CFG_USE_PIPES TRUE
Pipes APIs.
- #define CH_CFG_USE_OBJ_CACHES TRUE
Objects Caches APIs.
- #define CH_CFG_USE_DELEGATES TRUE
Delegate threads APIs.
- #define CH_CFG_USE_JOBS TRUE
Jobs Queues APIs.

Objects factory options

- `#define CH_CFG_USE_FACTORY TRUE`
Objects Factory APIs.
- `#define CH_CFG_FACTORY_MAX_NAMES_LENGTH 8`
Maximum length for object names.
- `#define CH_CFG_FACTORY_OBJECTS_REGISTRY TRUE`
Enables the registry of generic objects.
- `#define CH_CFG_FACTORY_GENERIC_BUFFERS TRUE`
Enables factory for generic buffers.
- `#define CH_CFG_FACTORY_SEMAPHORES TRUE`
Enables factory for semaphores.
- `#define CH_CFG_FACTORY_MAILBOXES TRUE`
Enables factory for mailboxes.
- `#define CH_CFG_FACTORY_OBJ_FIFOS TRUE`
Enables factory for objects FIFOs.
- `#define CH_CFG_FACTORY_PIPES TRUE`
Enables factory for Pipes.

Debug options

- `#define CH_DBG_STATISTICS FALSE`
Debug option, kernel statistics.
- `#define CH_DBG_SYSTEM_STATE_CHECK TRUE`
Debug option, system state check.
- `#define CH_DBG_ENABLE_CHECKS TRUE`
Debug option, parameters checks.
- `#define CH_DBG_ENABLE_ASSERTS TRUE`
Debug option, consistency checks.
- `#define CH_DBG_TRACE_MASK CH_DBG_TRACE_MASK_ALL`
Debug option, trace buffer.
- `#define CH_DBG_TRACE_BUFFER_SIZE 128`
Trace buffer entries.
- `#define CH_DBG_ENABLE_STACK_CHECK TRUE`
Debug option, stack checks.
- `#define CH_DBG_FILL_THREADS TRUE`
Debug option, stacks initialization.
- `#define CH_DBG_THREADS_PROFILING FALSE`
Debug option, threads profiling.

Kernel hooks

- `#define CH_CFG_SYSTEM_EXTRA_FIELDS /* Add system custom fields here. */`
System structure extension.
- `#define CH_CFG_SYSTEM_INIT_HOOK()`
System initialization hook.
- `#define CH_CFG_OS_INSTANCE_EXTRA_FIELDS /* Add OS instance custom fields here. */`
OS instance structure extension.
- `#define CH_CFG_OS_INSTANCE_INIT_HOOK(oip)`

- `#define CH_CFG_THREAD_EXTRA_FIELDS /* Add threads custom fields here.*/`
Threads descriptor structure extension.
- `#define CH_CFG_THREAD_INIT_HOOK(tp)`
Threads initialization hook.
- `#define CH_CFG_THREAD_EXIT_HOOK(tp)`
Threads finalization hook.
- `#define CH_CFG_CONTEXT_SWITCH_HOOK(ntp, otp)`
Context switch hook.
- `#define CH_CFG_IRQ_PROLOGUE_HOOK()`
ISR enter hook.
- `#define CH_CFG_IRQ_EPILOGUE_HOOK()`
ISR exit hook.
- `#define CH_CFG_IDLE_ENTER_HOOK()`
Idle thread enter hook.
- `#define CH_CFG_IDLE_LEAVE_HOOK()`
Idle thread leave hook.
- `#define CH_CFG_IDLE_LOOP_HOOK()`
Idle Loop hook.
- `#define CH_CFG_SYSTEM_TICK_HOOK()`
System tick event hook.
- `#define CH_CFG_SYSTEM_HALT_HOOK(reason)`
System halt hook.
- `#define CH_CFG_TRACE_HOOK(tep)`
Trace hook.
- `#define CH_CFG_RUNTIMEFAULTS_HOOK(mask)`
Runtime Faults Collection Unit hook.

7.8.2 Macro Definition Documentation

7.8.2.1 CH_CFG_SMP_MODE

```
#define CH_CFG_SMP_MODE FALSE
```

Handling of instances.

Note

If enabled then threads assigned to various instances can interact each other using the same synchronization objects. If disabled then each OS instance is a separate world, no direct interactions are handled by the OS.

7.8.2.2 CH_CFG_ST_RESOLUTION

```
#define CH_CFG_ST_RESOLUTION 32
```

System time counter resolution.

Note

Allowed values are 16, 32 or 64 bits.

7.8.2.3 CH_CFG_ST_FREQUENCY

```
#define CH_CFG_ST_FREQUENCY 10000
```

System tick frequency.

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

7.8.2.4 CH_CFG_INTERVALS_SIZE

```
#define CH_CFG_INTERVALS_SIZE 32
```

Time intervals data size.

Note

Allowed values are 16, 32 or 64 bits.

7.8.2.5 CH_CFG_TIME_TYPES_SIZE

```
#define CH_CFG_TIME_TYPES_SIZE 32
```

Time types data size.

Note

Allowed values are 16 or 32 bits.

7.8.2.6 CH_CFG_ST_TIMEDELTA

```
#define CH_CFG_ST_TIMEDELTA 2
```

Time delta constant for the tick-less mode.

Note

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

7.8.2.7 CH_CFG_TIME_QUANTUM

```
#define CH_CFG_TIME_QUANTUM 0
```

Round robin interval.

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

Note

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

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

7.8.2.8 CH_CFG_NO_IDLE_THREAD

```
#define CH_CFG_NO_IDLE_THREAD FALSE
```

Idle thread automatic spawn suppression.

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

7.8.2.9 CH_CFG_OPTIMIZE_SPEED

```
#define CH_CFG_OPTIMIZE_SPEED TRUE
```

OS optimization.

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

Note

This is not related to the compiler optimization options.

The default is `TRUE`.

7.8.2.10 CH_CFG_USE_TM

```
#define CH_CFG_USE_TM TRUE
```

Time Measurement APIs.

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

Note

The default is TRUE.

7.8.2.11 CH_CFG_USE_TIMESTAMP

```
#define CH_CFG_USE_TIMESTAMP TRUE
```

Time Stamps APIs.

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

Note

The default is TRUE.

7.8.2.12 CH_CFG_USE_REGISTRY

```
#define CH_CFG_USE_REGISTRY TRUE
```

Threads registry APIs.

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

Note

The default is TRUE.

7.8.2.13 CH_CFG_USE_WAITEXIT

```
#define CH_CFG_USE_WAITEXIT TRUE
```

Threads synchronization APIs.

If enabled then the [chThdWait\(\)](#) function is included in the kernel.

Note

The default is TRUE.

7.8.2.14 CH_CFG_USE_SEMAPHORES

```
#define CH_CFG_USE_SEMAPHORES TRUE
```

Semaphores APIs.

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

Note

The default is TRUE.

7.8.2.15 CH_CFG_USE_SEMAPHORES_PRIORITY

```
#define CH_CFG_USE_SEMAPHORES_PRIORITY FALSE
```

Semaphores queuing mode.

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

Note

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

Requires CH_CFG_USE_SEMAPHORES.

7.8.2.16 CH_CFG_USE_MUTEXES

```
#define CH_CFG_USE_MUTEXES TRUE
```

Mutexes APIs.

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

Note

The default is TRUE.

7.8.2.17 CH_CFG_USE_MUTEXES_RECURSIVE

```
#define CH_CFG_USE_MUTEXES_RECURSIVE FALSE
```

Enables recursive behavior on mutexes.

Note

Recursive mutexes are heavier and have an increased memory footprint.

The default is FALSE.

Requires CH_CFG_USE_MUTEXES.

7.8.2.18 CH_CFG_USE_CONDVARs

```
#define CH_CFG_USE_CONDVARs TRUE
```

Conditional Variables APIs.

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

Note

The default is TRUE.

Requires CH_CFG_USE_MUTEXES.

7.8.2.19 CH_CFG_USE_CONDVARs_TIMEOUT

```
#define CH_CFG_USE_CONDVARs_TIMEOUT TRUE
```

Conditional Variables APIs with timeout.

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

Note

The default is TRUE.

Requires CH_CFG_USE_CONDVARs.

7.8.2.20 CH_CFG_USE_EVENTS

```
#define CH_CFG_USE_EVENTS TRUE
```

Events Flags APIs.

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

Note

The default is TRUE.

7.8.2.21 CH_CFG_USE_EVENTS_TIMEOUT

```
#define CH_CFG_USE_EVENTS_TIMEOUT TRUE
```

Events Flags APIs with timeout.

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

Note

The default is TRUE.

Requires CH_CFG_USE_EVENTS.

7.8.2.22 CH_CFG_USE_MESSAGES

```
#define CH_CFG_USE_MESSAGES TRUE
```

Synchronous Messages APIs.

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

Note

The default is TRUE.

7.8.2.23 CH_CFG_USE_MESSAGES_PRIORITY

```
#define CH_CFG_USE_MESSAGES_PRIORITY FALSE
```

Synchronous Messages queuing mode.

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

Note

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

Requires CH_CFG_USE_MESSAGES.

7.8.2.24 CH_CFG_USE_DYNAMIC

```
#define CH_CFG_USE_DYNAMIC TRUE
```

Dynamic Threads APIs.

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

Note

The default is TRUE.

Requires CH_CFG_USE_WAITEXIT.

Requires CH_CFG_USE_HEAP and/or CH_CFG_USE_MEMPOOLS.

7.8.2.25 CH_CFG_USE_MAILBOXES

```
#define CH_CFG_USE_MAILBOXES TRUE
```

Mailboxes APIs.

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

Note

The default is TRUE.

Requires CH_CFG_USE_SEMAPHORES.

7.8.2.26 CH_CFG_USE_MEMCORE

```
#define CH_CFG_USE_MEMCORE TRUE
```

Core Memory Manager APIs.

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

Note

The default is TRUE.

7.8.2.27 CH_CFG_MEMCORE_SIZE

```
#define CH_CFG_MEMCORE_SIZE 0
```

Managed RAM size.

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

Note

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

Requires CH_CFG_USE_MEMCORE.

7.8.2.28 CH_CFG_USE_HEAP

```
#define CH_CFG_USE_HEAP TRUE
```

Heap Allocator APIs.

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

Note

The default is TRUE.

Requires CH_CFG_USE_MEMCORE and either CH_CFG_USE_MUTEXES or CH_CFG_USE_SEMAPHORES.

Mutexes are recommended.

7.8.2.29 CH_CFG_USE_MEMPOOLS

```
#define CH_CFG_USE_MEMPOOLS TRUE
```

Memory Pools Allocator APIs.

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

Note

The default is TRUE.

7.8.2.30 CH_CFG_USE_OBJ_FIFOS

```
#define CH_CFG_USE_OBJ_FIFOS TRUE
```

Objects FIFOs APIs.

If enabled then the objects FIFOs APIs are included in the kernel.

Note

The default is TRUE.

7.8.2.31 CH_CFG_USE_PIPES

```
#define CH_CFG_USE_PIPES TRUE
```

Pipes APIs.

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

Note

The default is TRUE.

7.8.2.32 CH_CFG_USE_OBJ_CACHES

```
#define CH_CFG_USE_OBJ_CACHES TRUE
```

Objects Caches APIs.

If enabled then the objects caches APIs are included in the kernel.

Note

The default is TRUE.

7.8.2.33 CH_CFG_USE_DELEGATES

```
#define CH_CFG_USE_DELEGATES TRUE
```

Delegate threads APIs.

If enabled then the delegate threads APIs are included in the kernel.

Note

The default is TRUE.

7.8.2.34 CH_CFG_USE_JOBS

```
#define CH_CFG_USE_JOBS TRUE
```

Jobs Queues APIs.

If enabled then the jobs queues APIs are included in the kernel.

Note

The default is TRUE.

7.8.2.35 CH_CFG_USE_FACTORY

```
#define CH_CFG_USE_FACTORY TRUE
```

Objects Factory APIs.

If enabled then the objects factory APIs are included in the kernel.

Note

The default is FALSE.

7.8.2.36 CH_CFG_FACTORY_MAX_NAMES_LENGTH

```
#define CH_CFG_FACTORY_MAX_NAMES_LENGTH 8
```

Maximum length for object names.

If the specified length is zero then the name is stored by pointer but this could have unintended side effects.

7.8.2.37 CH_CFG_FACTORY_OBJECTS_REGISTRY

```
#define CH_CFG_FACTORY_OBJECTS_REGISTRY TRUE
```

Enables the registry of generic objects.

7.8.2.38 CH_CFG_FACTORY_GENERIC_BUFFERS

```
#define CH_CFG_FACTORY_GENERIC_BUFFERS TRUE
```

Enables factory for generic buffers.

7.8.2.39 CH_CFG_FACTORY_SEMAPHORES

```
#define CH_CFG_FACTORY_SEMAPHORES TRUE
```

Enables factory for semaphores.

7.8.2.40 CH_CFG_FACTORY_MAILBOXES

```
#define CH_CFG_FACTORY_MAILBOXES TRUE
```

Enables factory for mailboxes.

7.8.2.41 CH_CFG_FACTORY_OBJ_FIFOS

```
#define CH_CFG_FACTORY_OBJ_FIFOS TRUE
```

Enables factory for objects FIFOs.

7.8.2.42 CH_CFG_FACTORY_PIPES

```
#define CH_CFG_FACTORY_PIPES TRUE
```

Enables factory for Pipes.

7.8.2.43 CH_DBG_STATISTICS

```
#define CH_DBG_STATISTICS FALSE
```

Debug option, kernel statistics.

Note

The default is FALSE.

7.8.2.44 CH_DBG_SYSTEM_STATE_CHECK

```
#define CH_DBG_SYSTEM_STATE_CHECK TRUE
```

Debug option, system state check.

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

Note

The default is FALSE.

7.8.2.45 CH_DBG_ENABLE_CHECKS

```
#define CH_DBG_ENABLE_CHECKS TRUE
```

Debug option, parameters checks.

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

Note

The default is FALSE.

7.8.2.46 CH_DBG_ENABLE_ASSERTS

```
#define CH_DBG_ENABLE_ASSERTS TRUE
```

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.

7.8.2.47 CH_DBG_TRACE_MASK

```
#define CH_DBG_TRACE_MASK CH_DBG_TRACE_MASK_ALL
```

Debug option, trace buffer.

If enabled then the trace buffer is activated.

Note

The default is CH_DBG_TRACE_MASK_DISABLED.

7.8.2.48 CH_DBG_TRACE_BUFFER_SIZE

```
#define CH_DBG_TRACE_BUFFER_SIZE 128
```

Trace buffer entries.

Note

The trace buffer is only allocated if CH_DBG_TRACE_MASK is different from CH_DBG_TRACE_MASK_DEFINED.

7.8.2.49 CH_DBG_ENABLE_STACK_CHECK

```
#define CH_DBG_ENABLE_STACK_CHECK TRUE
```

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.

7.8.2.50 CH_DBG_FILL_THREADS

```
#define CH_DBG_FILL_THREADS TRUE
```

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.

7.8.2.51 CH_DBG_THREADS_PROFILING

```
#define CH_DBG_THREADS_PROFILING FALSE
```

Debug option, threads profiling.

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

Note

The default is FALSE.

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

7.8.2.52 CH_CFG_SYSTEM_EXTRA_FIELDS

```
#define CH_CFG_SYSTEM_EXTRA_FIELDS /* Add system custom fields here.*/
```

System structure extension.

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

7.8.2.53 CH_CFG_SYSTEM_INIT_HOOK

```
#define CH_CFG_SYSTEM_INIT_HOOK( )
```

Value:

```
{ \
/* Add system initialization code here.*/ \
}
```

System initialization hook.

User initialization code added to the `chSysInit()` function just before interrupts are enabled globally.

7.8.2.54 CH_CFG_OS_INSTANCE_EXTRA_FIELDS

```
#define CH_CFG_OS_INSTANCE_EXTRA_FIELDS /* Add OS instance custom fields here.*/
```

OS instance structure extension.

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

7.8.2.55 CH_CFG_OS_INSTANCE_INIT_HOOK

```
#define CH_CFG_OS_INSTANCE_INIT_HOOK( \
oip )
```

Value:

```
{ \
/* Add OS instance initialization code here.*/ \
}
```

OS instance initialization hook.

Parameters

in	<i>oip</i>	pointer to the <code>os_instance_t</code> structure
----	------------	---

7.8.2.56 CH_CFG_THREAD_EXTRA_FIELDS

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

Threads descriptor structure extension.

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

7.8.2.57 CH_CFG_THREAD_INIT_HOOK

```
#define CH_CFG_THREAD_INIT_HOOK(
    tp )
```

Value:

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

Threads initialization hook.

User initialization code added to the `_thread_init()` function.

Note

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

Parameters

in	<i>tp</i>	pointer to the <code>thread_t</code> structure
----	-----------	--

7.8.2.58 CH_CFG_THREAD_EXIT_HOOK

```
#define CH_CFG_THREAD_EXIT_HOOK(
    tp )
```

Value:

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

Threads finalization hook.

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

Parameters

in	<i>tp</i>	pointer to the <code>thread_t</code> structure
----	-----------	--

7.8.2.59 CH_CFG_CONTEXT_SWITCH_HOOK

```
#define CH_CFG_CONTEXT_SWITCH_HOOK(
    ntp,
    otp )
```

Value:

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

Context switch hook.

This hook is invoked just before switching between threads.

Parameters

in	<i>ntp</i>	thread being switched in
in	<i>otp</i>	thread being switched out

7.8.2.60 CH_CFG_IRQ_PROLOGUE_HOOK

```
#define CH_CFG_IRQ_PROLOGUE_HOOK( )
```

Value:

```
{
    /* IRQ prologue code here.*/
}
```

ISR enter hook.

7.8.2.61 CH_CFG_IRQ_EPILOGUE_HOOK

```
#define CH_CFG_IRQ_EPILOGUE_HOOK( )
```

Value:

```
{
    /* IRQ epilogue code here.*/
}
```

ISR exit hook.

7.8.2.62 CH_CFG_IDLE_ENTER_HOOK

```
#define CH_CFG_IDLE_ENTER_HOOK( )
```

Value:

```
{ \
/* Idle-enter code here.*/ \
}
```

Idle thread enter hook.

Note

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

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

7.8.2.63 CH_CFG_IDLE_LEAVE_HOOK

```
#define CH_CFG_IDLE_LEAVE_HOOK( )
```

Value:

```
{ \
/* Idle-leave code here.*/ \
}
```

Idle thread leave hook.

Note

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

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

7.8.2.64 CH_CFG_IDLE_LOOP_HOOK

```
#define CH_CFG_IDLE_LOOP_HOOK( )
```

Value:

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

Idle Loop hook.

This hook is continuously invoked by the idle thread loop.

7.8.2.65 CH_CFG_SYSTEM_TICK_HOOK

```
#define CH_CFG_SYSTEM_TICK_HOOK( )
```

Value:

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

System tick event hook.

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

7.8.2.66 CH_CFG_SYSTEM_HALT_HOOK

```
#define CH_CFG_SYSTEM_HALT_HOOK(\n    reason )
```

Value:

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

System halt hook.

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

7.8.2.67 CH_CFG_TRACE_HOOK

```
#define CH_CFG_TRACE_HOOK(\n    tep )
```

Value:

```
{\n    /* Trace code here.*/\n}
```

Trace hook.

This hook is invoked each time a new record is written in the trace buffer.

7.8.2.68 CH_CFG_RUNTIMEFAULTS_HOOK

```
#define CH_CFG_RUNTIMEFAULTS_HOOK(\n    mask )
```

Value:

```
{\n    /* Faults handling code here.*/\n}
```

Runtime Faults Collection Unit hook.

This hook is invoked each time new faults are collected and stored.

7.9 Checks

This module performs a series of checks on configuration data, it is able to detect and reject obsolete or incomplete [chconf.h](#) files.

7.10 Restrictions

7.10.1 Detailed Description

This module is responsible for applying license-related restrictions to the configuration options.

7.11 System

7.11.1 Detailed Description

Modules

- Port Interface
- OS Types and Structures
- OS Instances
- Runtime Faults Collection Unit
- Lists and Queues
- Scheduler

7.12 Port Interface

7.12.1 Detailed Description

This module performs checks on the information exported by the port layer. The port layer is checked at compile time in order to make sure that it exports all the required macros and definitions.

Note

This module does not export any functionality.

7.13 OS Types and Structures

7.13.1 Detailed Description

Macros

- `#define __CH_STRINGIFY(a) #a`
Utility to make the parameter a quoted string.
- `#define __CH_OFFSETOF(st, m)`
Structure field offset utility.
- `#define __CH_USED(x) (void)(x)`
Marks an expression result as used.
- `#define likely(x) PORT_LIKELY(x)`
Marks a boolean expression as likely true.
- `#define unlikely(x) PORT_UNLIKELY(x)`
Marks a boolean expression as likely false.

Kernel types

- `typedef port_rtcnt_t rtcnt_t`
- `typedef port_rftime_t rftime_t`
- `typedef port_syssts_t syssts_t`
- `typedef port_stkalign_t stkalign_t`
- `typedef uint8_t tmode_t`
- `typedef uint8_t tstate_t`
- `typedef uint8_t trefs_t`
- `typedef uint8_t tslices_t`
- `typedef uint32_t tprio_t`
- `typedef int32_t msg_t`
- `typedef int32_t eventid_t`
- `typedef uint32_t eventmask_t`
- `typedef uint32_t eventflags_t`
- `typedef int32_t cnt_t`
- `typedef uint32_t ucnt_t`

Typedefs

- `typedef unsigned core_id_t`
Type of a core identifier.
- `typedef struct ch_thread thread_t`
Type of a thread structure.
- `typedef struct ch_os_instance os_instance_t`
Type of an OS instance structure.
- `typedef struct ch_virtual_timer virtual_timer_t`
Type of a Virtual Timer.
- `typedef void(* vfunc_t) (virtual_timer_t *vtp, void *p)`
Type of a Virtual Timer callback function.
- `typedef struct ch_virtual_timers_list virtual_timers_list_t`
Type of virtual timers list header.
- `typedef struct ch_registry registry_t`

- `typedef thread_t * thread_reference_t`
Type of a thread reference.
- `typedef struct ch_threads_queue threads_queue_t`
Type of a threads queue.
- `typedef struct ch_ready_list ready_list_t`
Type of a ready list header.
- `typedef struct ch_os_instance_config os_instance_config_t`
Type of an system instance configuration.
- `typedef struct ch_system ch_system_t`
Type of system data structure.

Data Structures

- `struct ch_virtual_timer`
Structure representing a Virtual Timer.
- `struct ch_virtual_timers_list`
Type of virtual timers list header.
- `struct ch_registry`
Type of a registry structure.
- `struct ch_threads_queue`
Type of a threads queue.
- `struct ch_thread`
Structure representing a thread.
- `struct ch_ready_list`
Type of a ready list header.
- `struct ch_os_instance_config`
Type of an system instance configuration.
- `struct ch_os_instance`
System instance data structure.
- `struct ch_system`
Type of system data structure.

Functions

- `void chSysHalt (const char *reason)`
Halts the system.

Enumerations

- `enum system_state_t`
Global state of the operating system.

7.13.2 Macro Definition Documentation

7.13.2.1 __CH_STRINGIFY

```
#define __CH_STRINGIFY(
```

$$\quad \quad \quad a \) \ #a$$

Utility to make the parameter a quoted string.

Parameters

in	<i>a</i>	literal to be string-ified
----	----------	----------------------------

7.13.2.2 __CH_OFFSETOF

```
#define __CH_OFFSETOF(
    st,
    m )
```

Value:

```
/*lint -save -e9005 -e9033 -e413 [11.8, 10.8, 1.3] Normal pointers
   arithmetic, it is safe.*/
((size_t)((char *)(&(st *0))->m - (char *)0)) \
/*lint -restore*/
```

Structure field offset utility.

Parameters

in	<i>st</i>	structured type name
in	<i>m</i>	field name in the structured type

Returns

The offset of the field in the structured type.

7.13.2.3 __CH_USED

```
#define __CH_USED(
    x ) (void)(x)
```

Marks an expression result as used.

Parameters

in	<i>x</i>	a valid expression
----	----------	--------------------

7.13.2.4 likely

```
#define likely(
    x ) PORT_LIKELY(x)
```

Marks a boolean expression as likely true.

Note

No namespace prefix for this macro because it is commonly defined by operating systems.

Parameters

in	x	a valid expression
----	---	--------------------

7.13.2.5 unlikely

```
#define unlikely(  
    x ) PORT_UNLIKELY(x)
```

Marks a boolean expression as likely false.

Note

No namespace prefix for this macro because it is commonly defined by operating systems.

Parameters

in	x	a valid expression
----	---	--------------------

7.13.3 Typedef Documentation**7.13.3.1 rtcnt_t**

```
typedef port_rtcnt_t rtcnt_t
```

Realtime counter.

7.13.3.2 rttimer_t

```
typedef port_rttimer_t rttimer_t
```

Realtime accumulator.

7.13.3.3 syssts_t

```
typedef port_syssts_t syssts_t
```

System status word.

7.13.3.4 `stkalign_t`

```
typedef port_stkalign_t stkalign_t
```

Stack alignment type.

7.13.3.5 `tmode_t`

```
typedef uint8_t tmode_t
```

Thread flags.

7.13.3.6 `tstate_t`

```
typedef uint8_t tstate_t
```

Thread state.

7.13.3.7 `trrefs_t`

```
typedef uint8_t trrefs_t
```

Thread references counter.

7.13.3.8 `tslices_t`

```
typedef uint8_t tslices_t
```

Thread time slices counter.

7.13.3.9 `tprio_t`

```
typedef uint32_t tprio_t
```

Thread priority.

7.13.3.10 `msg_t`

```
typedef int32_t msg_t
```

Inter-thread message.

7.13.3.11 eventid_t

```
typedef int32_t eventid_t
```

Numeric event identifier.

7.13.3.12 eventmask_t

```
typedef uint32_t eventmask_t
```

Mask of event identifiers.

7.13.3.13 eventflags_t

```
typedef uint32_t eventflags_t
```

Mask of event flags.

7.13.3.14 cnt_t

```
typedef int32_t cnt_t
```

Generic signed counter.

7.13.3.15 ucnt_t

```
typedef uint32_t ucnt_t
```

Generic unsigned counter.

7.13.3.16 core_id_t

```
typedef unsigned core_id_t
```

Type of a core identifier.

Note

Core identifiers have ranges from 0 to PORT_CORES_NUMBER - 1.

7.13.3.17 `thread_t`

```
typedef struct ch_thread thread_t
```

Type of a thread structure.

7.13.3.18 `os_instance_t`

```
typedef struct ch_os_instance os_instance_t
```

Type of an OS instance structure.

7.13.3.19 `virtual_timer_t`

```
typedef struct ch_virtual_timer virtual_timer_t
```

Type of a Virtual Timer.

7.13.3.20 `vfunc_t`

```
typedef void(* vfunc_t) (virtual_timer_t *vtp, void *p)
```

Type of a Virtual Timer callback function.

Parameters

<code>in</code>	<code>vtp</code>	pointer to the <code>virtual_timer_t</code> calling this callback
	<code>[in]</code>	p optional argument to the callback

Returns

The interval to be reloaded into the timer or zero.

Return values

<code>0</code>	if the timer must not be reloaded.
----------------	------------------------------------

7.13.3.21 `virtual_timers_list_t`

```
typedef struct ch_virtual_timers_list virtual_timers_list_t
```

Type of virtual timers list header.

Note

The timers list is implemented as a double link bidirectional list in order to make the unlink time constant, the reset of a virtual timer is often used in the code.

7.13.3.22 `registry_t`

```
typedef struct ch_registry registry_t
```

Type of a registry structure.

7.13.3.23 `thread_reference_t`

```
typedef thread_t* thread_reference_t
```

Type of a thread reference.

7.13.3.24 `threads_queue_t`

```
typedef struct ch_threads_queue threads_queue_t
```

Type of a threads queue.

7.13.3.25 `ready_list_t`

```
typedef struct ch_ready_list ready_list_t
```

Type of a ready list header.

7.13.3.26 `os_instance_config_t`

```
typedef struct ch_os_instance_config os_instance_config_t
```

Type of an system instance configuration.

7.13.3.27 ch_system_t

```
typedef struct ch_system ch_system_t
```

Type of system data structure.

7.13.4 Enumeration Type Documentation

7.13.4.1 system_state_t

```
enum system_state_t
```

Global state of the operating system.

7.13.5 Function Documentation

7.13.5.1 chSysHalt()

```
void chSysHalt (
    const char * reason )
```

Halts the system.

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

Note

Can be invoked from any system state.

Parameters

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

Function Class:

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

Here is the call graph for this function:



7.14 OS Instances

7.14.1 Detailed Description

OS instances management.

Macros

- `#define __instance_get_currthread(oip) (oip)->rlist.current`
Current thread pointer get macro.
- `#define __instance_set_currthread(oip, tp) (oip)->rlist.current = (tp)`
Current thread pointer set macro.

Functions

- `static void __idle_thread (void *p)`
This function implements the idle thread infinite loop.
- `void chInstanceObjectInit (os_instance_t *oip, const os_instance_config_t *oicp)`
Initializes a system instance.

7.14.2 Macro Definition Documentation

7.14.2.1 __instance_get_currthread

```
#define __instance_get_currthread(
    oip ) (oip)->rlist.current
```

Current thread pointer get macro.

Note

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

7.14.2.2 __instance_set_currthread

```
#define __instance_set_currthread(
    oip,
    tp ) (oip)->rlist.current = (tp)
```

Current thread pointer set macro.

7.14.3 Function Documentation

7.14.3.1 __idle_thread()

```
static void __idle_thread (
    void * p ) [static]
```

This function implements the idle thread infinite loop.

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

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

Parameters

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

7.14.3.2 chInstanceObjectInit()

```
void chInstanceObjectInit (
    os_instance_t * oip,
    const os_instance_config_t * oicp )
```

Initializes a system instance.

Note

The system instance is in I-Lock state after initialization.

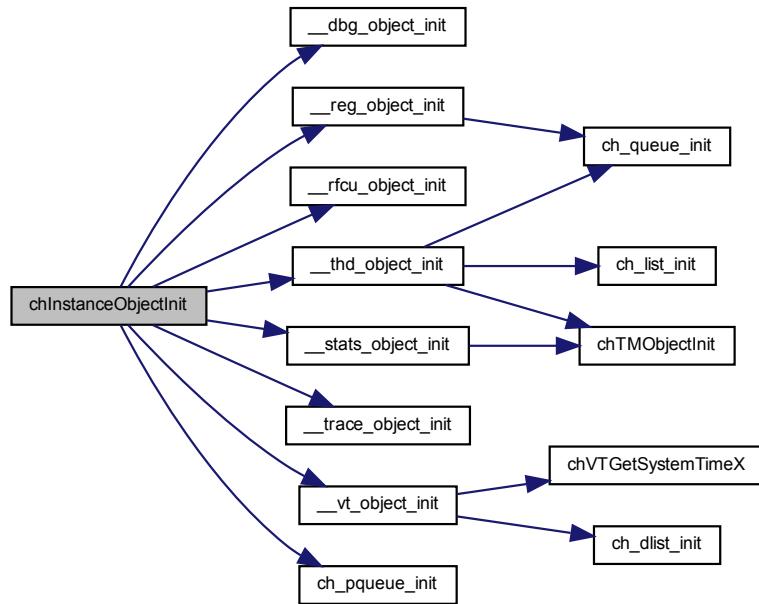
Parameters

out	<i>oip</i>	pointer to the <code>os_instance_t</code> structure
in	<i>oicp</i>	pointer to the <code>os_instance_config_t</code> structure

Function Class:

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

Here is the call graph for this function:



7.15 Runtime Faults Collection Unit

7.15.1 Detailed Description

Runtime Faults Collection Unit service.

Macros

- `#define CH_RFCU_ALLFAULTS ((rfcu_mask_t)-1)`
Mask of all faults.

Predefined Faults

- `#define CH_RFCU_VT_INSUFFICIENT_DELTA 1U`
- `#define CH_RFCU_VT_SKIPPED_DEADLINE 2U`

Typedefs

- `typedef uint32_t rfcu_mask_t`
Type of a faults mask.
- `typedef struct ch_rfcu rfcu_t`
Type of an RFCU structure.

Data Structures

- `struct ch_rfcu`
Type of an RFCU structure.

Functions

- `void chRFCUCollectFaults1 (rfcu_mask_t mask)`
Adds fault flags to the current mask.
- `rfcu_mask_t chRFCUGetAndClearFaults1 (rfcu_mask_t mask)`
Returns the current faults mask clearing it.
- `static void __rfcu_object_init (rfcu_t *rfcup)`
Runtime Faults Collection Unit initialization.

7.15.2 Macro Definition Documentation

7.15.2.1 CH_RFCU_ALLFAULTS

```
#define CH_RFCU_ALLFAULTS ((rfcu_mask_t)-1)
```

Mask of all faults.

7.15.3 Typedef Documentation

7.15.3.1 rfcu_mask_t

```
typedef uint32_t rfcu_mask_t
```

Type of a faults mask.

7.15.3.2 rfcu_t

```
typedef struct ch_rfcu rfcu_t
```

Type of an RFCU structure.

7.15.4 Function Documentation

7.15.4.1 chRFCUCollectFaultsI()

```
void chRFCUCollectFaultsI (
    rfcu_mask_t mask )
```

Adds fault flags to the current mask.

Parameters

in	mask	fault flags to be added
----	------	-------------------------

7.15.4.2 chRFCUGetAndClearFaultsI()

```
rfcu_mask_t chRFCUGetAndClearFaultsI (
    rfcu_mask_t mask )
```

Returns the current faults mask clearing it.

Parameters

in	mask	mask of faults to be read and cleared
----	------	---------------------------------------

Returns

The current faults mask.

Return values

0	if no faults were collected since last call to this function.
---	---

7.15.4.3 __rfcu_object_init()

```
static void __rfcu_object_init (
    rfcu_t * rfcup ) [inline], [static]
```

Runtime Faults Collection Unit initialization.

Note

Internal use only.

Parameters

out	rfcup	pointer to the rfcu_t structure
-----	-------	---------------------------------

Function Class:

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

7.16 Lists and Queues

7.16.1 Detailed Description

Macros

- `#define __CH_QUEUE_DATA(name) {(<ch_queue_t *>)&name, (<ch_queue_t *>)&name}`
Data part of a static queue object initializer.
- `#define CH_QUEUE_DECL(name) ch_queue_t name = __CH_QUEUE_DATA(name)`
Static queue object initializer.

Typedefs

- `typedef struct ch_list ch_list_t`
Type of a generic single link list header and element.
- `typedef struct ch_queue ch_queue_t`
Type of a generic bidirectional linked list header and element.
- `typedef struct ch_priority_queue ch_priority_queue_t`
Type of a generic priority-ordered bidirectional linked list header and element.
- `typedef struct ch_delta_list ch_delta_list_t`
Type of a generic bidirectional linked delta list header and element.

Data Structures

- `struct ch_list`
Structure representing a generic single link list header and element.
- `struct ch_queue`
Structure representing a generic bidirectional linked list header and element.
- `struct ch_priority_queue`
Structure representing a generic priority-ordered bidirectional linked list header and element.
- `struct ch_delta_list`
Delta list element and header structure.

Functions

- `static void ch_list_init (ch_list_t *lp)`
List initialization.
- `static bool ch_list_isempty (ch_list_t *lp)`
Evaluates to true if the specified list is empty.
- `static bool ch_list_notempty (ch_list_t *lp)`
Evaluates to true if the specified list is not empty.
- `static void ch_list_link (ch_list_t *lp, ch_list_t *p)`
Pushes an element on top of a stack list.
- `static ch_list_t * ch_list_unlink (ch_list_t *lp)`
Pops an element from the top of a stack list and returns it.
- `static void ch_queue_init (ch_queue_t *qp)`
Queue initialization.
- `static bool ch_queue_isempty (const ch_queue_t *qp)`

- static bool `ch_queue_notempty` (const `ch_queue_t` *qp)

Evaluates to true if the specified queue is empty.
- static void `ch_queue_insert` (`ch_queue_t` *qp, `ch_queue_t` *p)

Inserts an element into a queue.
- static `ch_queue_t` * `ch_queue_fifo_remove` (`ch_queue_t` *qp)

Removes the first-out element from a queue and returns it.
- static `ch_queue_t` * `ch_queue_lifo_remove` (`ch_queue_t` *qp)

Removes the last-out element from a queue and returns it.
- static `ch_queue_t` * `ch_queue_dequeue` (`ch_queue_t` *p)

Removes an element from a queue and returns it.
- static void `ch_pqueue_init` (`ch_priority_queue_t` *pqp)

Priority queue initialization.
- static `ch_priority_queue_t` * `ch_pqueue_remove_highest` (`ch_priority_queue_t` *pqp)

Removes the highest priority element from a priority queue and returns it.
- static `ch_priority_queue_t` * `ch_pqueue_insert_behind` (`ch_priority_queue_t` *pqp, `ch_priority_queue_t` *p)

Inserts an element in the priority queue placing it behind its peers.
- static `ch_priority_queue_t` * `ch_pqueue_insert_ahead` (`ch_priority_queue_t` *pqp, `ch_priority_queue_t` *p)

Inserts an element in the priority queue placing it ahead of its peers.
- static void `ch_dlist_init` (`ch_delta_list_t` *dlhp)

Delta list initialization.
- static bool `ch_dlist_isempty` (`ch_delta_list_t` *dlhp)

Evaluates to true if the specified delta list is empty.
- static bool `ch_dlist_notempty` (`ch_delta_list_t` *dlhp)

Evaluates to true if the specified queue is not empty.
- static bool `ch_dlist_islast` (`ch_delta_list_t` *dlhp, `ch_delta_list_t` *dlp)

Last element in the delta list check.
- static bool `ch_dlist_isfirst` (`ch_delta_list_t` *dlhp, `ch_delta_list_t` *dlp)

Fist element in the delta list check.
- static void `ch_dlist_insert_after` (`ch_delta_list_t` *dlhp, `ch_delta_list_t` *dlp, `sysinterval_t` delta)

Inserts an element after another header element.
- static void `ch_dlist_insert_before` (`ch_delta_list_t` *dlhp, `ch_delta_list_t` *dlp, `sysinterval_t` delta)

Inserts an element before another header element.
- static void `ch_dlist_insert` (`ch_delta_list_t` *dlhp, `ch_delta_list_t` *dlep, `sysinterval_t` delta)

Inserts an element in a delta list.
- static `ch_delta_list_t` * `ch_dlist_remove_first` (`ch_delta_list_t` *dlhp)

Dequeues an element from the delta list.
- static `ch_delta_list_t` * `ch_dlist_dequeue` (`ch_delta_list_t` *dlp)

Dequeues an element from the delta list.

7.16.2 Macro Definition Documentation

7.16.2.1 __CH_QUEUE_DATA

```
#define __CH_QUEUE_DATA(
    name ) { (ch_queue_t *) &name, (ch_queue_t *) &name }
```

Data part of a static queue object initializer.

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

Parameters

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

7.16.2.2 CH_QUEUE_DECL

```
#define CH_QUEUE_DECL( name ) ch_queue_t name = __CH_QUEUE_DATA(name)
```

Static queue object initializer.

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

Parameters

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

7.16.3 Typedef Documentation**7.16.3.1 ch_list_t**

```
typedef struct ch_list ch_list_t
```

Type of a generic single link list header and element.

7.16.3.2 ch_queue_t

```
typedef struct ch_queue ch_queue_t
```

Type of a generic bidirectional linked list header and element.

7.16.3.3 ch_priority_queue_t

```
typedef struct ch_priority_queue ch_priority_queue_t
```

Type of a generic priority-ordered bidirectional linked list header and element.

7.16.3.4 ch_delta_list_t

```
typedef struct ch_delta_list ch_delta_list_t
```

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

7.16.4 Function Documentation

7.16.4.1 ch_list_init()

```
static void ch_list_init (
    ch_list_t * lp ) [inline], [static]
```

List initialization.

Parameters

out	lp	pointer to the list header
-----	----	----------------------------

Function Class:

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

7.16.4.2 ch_list_isempty()

```
static bool ch_list_isempty (
    ch_list_t * lp ) [inline], [static]
```

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

Parameters

in	lp	pointer to the list header
----	----	----------------------------

Returns

The status of the list.

Function Class:

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

7.16.4.3 ch_list_notempty()

```
static bool ch_list_notempty (
    ch_list_t * lp ) [inline], [static]
```

Evaluates to true if the specified list is not empty.

Parameters

in	<i>lp</i>	pointer to the list header
----	-----------	----------------------------

Returns

The status of the list.

Function Class:

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

7.16.4.4 ch_list_link()

```
static void ch_list_link (
    ch_list_t * lp,
    ch_list_t * p ) [inline], [static]
```

Pushes an element on top of a stack list.

Parameters

in	<i>lp</i>	the pointer to the list header
in	<i>p</i>	the pointer to the element to be inserted in the list

Function Class:

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

7.16.4.5 ch_list_unlink()

```
static ch_list_t* ch_list_unlink (
    ch_list_t * lp ) [inline], [static]
```

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

Precondition

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

Parameters

in	<i>lp</i>	the pointer to the list header
----	-----------	--------------------------------

Returns

The removed element pointer.

Function Class:

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

7.16.4.6 ch_queue_init()

```
static void ch_queue_init (
    ch_queue_t * qp ) [inline], [static]
```

Queue initialization.

Parameters

out	<i>qp</i>	pointer to the queue header
-----	-----------	-----------------------------

Function Class:

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

7.16.4.7 ch_queue_isempty()

```
static bool ch_queue_isempty (
    const ch_queue_t * qp ) [inline], [static]
```

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

Parameters

in	<i>qp</i>	pointer to the queue header
----	-----------	-----------------------------

Returns

The status of the queue.

Function Class:

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

7.16.4.8 ch_queue_notempty()

```
static bool ch_queue_notempty (
    const ch_queue_t * qp ) [inline], [static]
```

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

Parameters

in	<i>qp</i>	pointer to the queue header
----	-----------	-----------------------------

Returns

The status of the queue.

Function Class:

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

7.16.4.9 ch_queue_insert()

```
static void ch_queue_insert (
    ch_queue_t * qp,
    ch_queue_t * p ) [inline], [static]
```

Inserts an element into a queue.

Parameters

in	<i>qp</i>	the pointer to the queue header
in	<i>p</i>	the pointer to the element to be inserted in the queue

Function Class:

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

7.16.4.10 ch_queue_fifo_remove()

```
static ch_queue_t* ch_queue_fifo_remove (
    ch_queue_t * qp ) [inline], [static]
```

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

Note

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

Parameters

in	<i>qp</i>	the pointer to the queue list header
----	-----------	--------------------------------------

Returns

The removed element pointer.

Function Class:

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

7.16.4.11 ch_queue_lifo_remove()

```
static ch_queue_t* ch_queue_lifo_remove (
    ch_queue_t * qp ) [inline], [static]
```

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

Note

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

Parameters

in	<i>qp</i>	the pointer to the queue list header
----	-----------	--------------------------------------

Returns

The removed element pointer.

Function Class:

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

7.16.4.12 ch_queue_dequeue()

```
static ch_queue_t* ch_queue_dequeue (
    ch_queue_t * p ) [inline], [static]
```

Removes an element from a queue and returns it.

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

Parameters

in	<i>p</i>	the pointer to the element to be removed from the queue
----	----------	---

Returns

The removed element pointer.

Function Class:

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

7.16.4.13 ch_pqueue_init()

```
static void ch_pqueue_init (
    ch_priority_queue_t * pqp ) [inline], [static]
```

Priority queue initialization.

Note

The queue header priority is initialized to zero, all other elements in the queue are assumed to have priority greater than zero.

Parameters

out	<i>pqp</i>	pointer to the priority queue header
-----	------------	--------------------------------------

Function Class:

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

7.16.4.14 ch_pqueue_remove_highest()

```
static ch_priority_queue_t* ch_pqueue_remove_highest (
    ch_priority_queue_t * pqp ) [inline], [static]
```

Removes the highest priority element from a priority queue and returns it.

Parameters

in	<i>pqp</i>	the pointer to the priority queue list header
----	------------	---

Returns

The removed element pointer.

Function Class:

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

7.16.4.15 ch_pqueue_insert_behind()

```
static ch_priority_queue_t* ch_pqueue_insert_behind (
    ch_priority_queue_t * pqp,
    ch_priority_queue_t * p ) [inline], [static]
```

Inserts an element in the priority queue placing it behind its peers.

The element is positioned behind all elements with higher or equal priority.

Parameters

in	<i>pqp</i>	the pointer to the priority queue list header
in	<i>p</i>	the pointer to the element to be inserted in the queue

Returns

The inserted element pointer.

Function Class:

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

7.16.4.16 ch_pqueue_insert_ahead()

```
static ch_priority_queue_t* ch_pqueue_insert_ahead (
    ch_priority_queue_t * pqp,
    ch_priority_queue_t * p ) [inline], [static]
```

Inserts an element in the priority queue placing it ahead of its peers.

The element is positioned ahead of all elements with higher or equal priority.

Parameters

in	<i>pqp</i>	the pointer to the priority queue list header
in	<i>p</i>	the pointer to the element to be inserted in the queue

Returns

The inserted element pointer.

Function Class:

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

7.16.4.17 ch_dlist_init()

```
static void ch_dlist_init (
    ch_delta_list_t * dlhp ) [inline], [static]
```

Delta list initialization.

Parameters

out	<i>dlhp</i>	pointer to the delta list header
-----	-------------	----------------------------------

Function Class:

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

7.16.4.18 ch_dlist_isempty()

```
static bool ch_dlist_isempty (
    ch_delta_list_t * dlhp ) [inline], [static]
```

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

Parameters

in	<i>dlhp</i>	pointer to the delta list header
----	-------------	----------------------------------

Returns

The status of the delta list.

Function Class:

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

7.16.4.19 ch_dlist_notempty()

```
static bool ch_dlist_notempty (
    ch_delta_list_t * dlhp ) [inline], [static]
```

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

Parameters

in	<i>dlhp</i>	pointer to the delta list header
----	-------------	----------------------------------

Returns

The status of the delta list.

Function Class:

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

7.16.4.20 ch_dlist_islast()

```
static bool ch_dlist_islast (
    ch_delta_list_t * dlhp,
    ch_delta_list_t * dlip ) [inline], [static]
```

Last element in the delta list check.

Parameters

in	<i>dlhp</i>	pointer to the delta list header
in	<i>dlip</i>	pointer to the delta list element

Function Class:

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

7.16.4.21 ch_dlist_isfirst()

```
static bool ch_dlist_isfirst (
```

```
ch_delta_list_t * dlhp,
ch_delta_list_t * dlp ) [inline], [static]
```

Fist element in the delta list check.

Parameters

in	<i>dlhp</i>	pointer to the delta list header
in	<i>dlp</i>	pointer to the delta list element

Function Class:

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

7.16.4.22 ch_dlist_insert_after()

```
static void ch_dlist_insert_after (
    ch_delta_list_t * dlhp,
    ch_delta_list_t * dlp,
    sysinterval_t delta ) [inline], [static]
```

Inserts an element after another header element.

Parameters

in	<i>dlhp</i>	pointer to the delta list header element
in	<i>dlp</i>	element to be inserted after the header element
in	<i>delta</i>	delta of the element to be inserted

Function Class:

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

7.16.4.23 ch_dlist_insert_before()

```
static void ch_dlist_insert_before (
    ch_delta_list_t * dlhp,
    ch_delta_list_t * dlp,
    sysinterval_t delta ) [inline], [static]
```

Inserts an element before another header element.

Parameters

in	<i>dlhp</i>	pointer to the delta list header element
in	<i>dlp</i>	element to be inserted before the header element
in	<i>delta</i>	delta of the element to be inserted

Function Class:

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

7.16.4.24 ch_dlist_insert()

```
static void ch_dlist_insert (
    ch_delta_list_t * dlhp,
    ch_delta_list_t * dlep,
    sysinterval_t delta ) [inline], [static]
```

Inserts an element in a delta list.

Parameters

in	<i>dlhp</i>	pointer to the delta list header element
in	<i>dlep</i>	element to be inserted before the header element
in	<i>delta</i>	delta of the element to be inserted

Function Class:

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

Here is the call graph for this function:

**7.16.4.25 ch_dlist_remove_first()**

```
static ch_delta_list_t* ch_dlist_remove_first (
    ch_delta_list_t * dlhp ) [inline], [static]
```

Dequeues an element from the delta list.

Parameters

in	<i>dlhp</i>	pointer to the delta list header
----	-------------	----------------------------------

Function Class:

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

7.16.4.26 ch_dlist_dequeue()

```
static ch_delta_list_t* ch_dlist_dequeue (
    ch_delta_list_t * dlp ) [inline], [static]
```

Dequeues an element from the delta list.

Parameters

in	<i>dlp</i>	pointer to the delta list element
----	------------	-----------------------------------

Function Class:

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

7.17 Scheduler

7.17.1 Detailed Description

This module provides the default portable scheduler code.

Macros

- `#define firstprio(rlp) ((rlp)->next->prio)`
Returns the priority of the first thread on the given ready list.
- `#define __sch_get_currthread() __instance_get_currthread(currcore)`
Current thread pointer get macro.

Wakeup status codes

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

Priority constants

- `#define NOPRIO (tprio_t)0`
Ready list header priority.
- `#define IDLEPRIO (tprio_t)1`
Idle priority.
- `#define LOWPRIO (tprio_t)2`
Lowest priority.
- `#define NORMALPRIO (tprio_t)128`
Normal priority.
- `#define HIGHPRIO (tprio_t)255`
Highest priority.

Thread states

- `#define CH_STATE_READY (tstate_t)0`
Waiting on the ready list.
- `#define CH_STATE_CURRENT (tstate_t)1`
Currently running.
- `#define CH_STATE_WTSTART (tstate_t)2`
Just created.
- `#define CH_STATE_SUSPENDED (tstate_t)3`
Suspended state.
- `#define CH_STATE_QUEUED (tstate_t)4`
On a queue.
- `#define CH_STATE_WTSEM (tstate_t)5`
On a semaphore.
- `#define CH_STATE_WTMutex (tstate_t)6`
On a mutex.
- `#define CH_STATE_WTCOND (tstate_t)7`
On a cond.variable.
- `#define CH_STATE_SLEEPING (tstate_t)8`
Sleeping.
- `#define CH_STATE_WTEXIT (tstate_t)9`
Waiting a thread.
- `#define CH_STATE_WTOREVT (tstate_t)10`
One event.
- `#define CH_STATE_WTANDEVT (tstate_t)11`
Several events.
- `#define CH_STATE SNDMSGQ (tstate_t)12`
Sending a message, in queue.
- `#define CH_STATE SNDMSG (tstate_t)13`
Sent a message, waiting answer.
- `#define CH_STATE_WTMSG (tstate_t)14`
Waiting for a message.
- `#define CH_STATE_FINAL (tstate_t)15`
Thread terminated.
- `#define CH_STATE_NAMES`
Thread states as array of strings.

Thread flags and attributes

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

Functions

- static `thread_t * __sch_ready_behind (thread_t *tp)`
Inserts a thread in the Ready List placing it behind its peers.
- static `thread_t * __sch_ready_ahead (thread_t *tp)`
Inserts a thread in the Ready List placing it ahead its peers.
- static void `__sch_reschedule_behind (void)`
Switches to the first thread on the runnable queue.
- static void `__sch_reschedule_ahead (void)`
Switches to the first thread on the runnable queue.
- void `ch_sch_prio_insert (ch_queue_t *qp, ch_queue_t *tp)`
Inserts a thread into a priority ordered queue.
- `thread_t * chSchReadyI (thread_t *tp)`
Inserts a thread in the Ready List placing it behind its peers.
- void `chSchGoSleepS (tstate_t newstate)`
Puts the current thread to sleep into the specified state.
- `msg_t chSchGoSleepTimeoutS (tstate_t newstate, sysinterval_t timeout)`
Puts the current thread to sleep into the specified state with timeout specification.
- void `chSchWakeUpS (thread_t *ntp, msg_t msg)`
Wakes up a thread.
- void `chSchRescheduleS (void)`
Performs a reschedule if a higher priority thread is runnable.
- bool `chSchIsPreemptionRequired (void)`
Evaluates if preemption is required.
- void `chSchDoPreemption (void)`
Switches to the first thread on the runnable queue.
- void `chSchPreemption (void)`
All-in-one preemption code.
- void `chSchDoYieldS (void)`
Yields the time slot.
- `thread_t * chSchSelectFirstI (void)`
Makes runnable the fist thread in the ready list, does not reschedule internally.

7.17.2 Macro Definition Documentation

7.17.2.1 MSG_OK

```
#define MSG_OK (msg_t) 0
```

Normal wakeup message.

7.17.2.2 MSG_TIMEOUT

```
#define MSG_TIMEOUT (msg_t)-1
```

Wakeup caused by a timeout condition.

7.17.2.3 MSG_RESET

```
#define MSG_RESET (msg_t)-2
```

Wakeup caused by a reset condition.

7.17.2.4 NOPRIO

```
#define NOPRIO (tprio_t) 0
```

Ready list header priority.

7.17.2.5 IDLEPRIO

```
#define IDLEPRIO (tprio_t) 1
```

Idle priority.

7.17.2.6 LOWPRIO

```
#define LOWPRIO (tprio_t) 2
```

Lowest priority.

7.17.2.7 NORMALPRIORITY

```
#define NORMALPRIORITY (tprio_t) 128
```

Normal priority.

7.17.2.8 HIGHPRIORITY

```
#define HIGHPRIORITY (tprio_t) 255
```

Highest priority.

7.17.2.9 CH_STATE_READY

```
#define CH_STATE_READY (tstate_t) 0
```

Waiting on the ready list.

7.17.2.10 CH_STATE_CURRENT

```
#define CH_STATE_CURRENT (tstate_t) 1
```

Currently running.

7.17.2.11 CH_STATE_WTSTART

```
#define CH_STATE_WTSTART (tstate_t) 2
```

Just created.

7.17.2.12 CH_STATE_SUSPENDED

```
#define CH_STATE_SUSPENDED (tstate_t) 3
```

Suspended state.

7.17.2.13 CH_STATE_QUEUED

```
#define CH_STATE_QUEUED (tstate_t) 4
```

On a queue.

7.17.2.14 CH_STATE_WTSEM

```
#define CH_STATE_WTSEM (tstate_t) 5
```

On a semaphore.

7.17.2.15 CH_STATE_WTMTX

```
#define CH_STATE_WTMTX (tstate_t) 6
```

On a mutex.

7.17.2.16 CH_STATE_WTCOND

```
#define CH_STATE_WTCOND (tstate_t) 7
```

On a cond.variable.

7.17.2.17 CH_STATE_SLEEPING

```
#define CH_STATE_SLEEPING (tstate_t) 8
```

Sleeping.

7.17.2.18 CH_STATE_WTEXIT

```
#define CH_STATE_WTEXIT (tstate_t)9
```

Waiting a thread.

7.17.2.19 CH_STATE_WTOREVT

```
#define CH_STATE_WTOREVT (tstate_t)10
```

One event.

7.17.2.20 CH_STATE_WTANDEVT

```
#define CH_STATE_WTANDEVT (tstate_t)11
```

Several events.

7.17.2.21 CH_STATE SNDMSGQ

```
#define CH_STATE SNDMSGQ (tstate_t)12
```

Sending a message, in queue.

7.17.2.22 CH_STATE SNDMSG

```
#define CH_STATE SNDMSG (tstate_t)13
```

Sent a message, waiting answer.

7.17.2.23 CH_STATE_WTMSG

```
#define CH_STATE_WTMSG (tstate_t)14
```

Waiting for a message.

7.17.2.24 CH_STATE_FINAL

```
#define CH_STATE_FINAL (tstate_t)15
```

Thread terminated.

7.17.2.25 CH_STATE_NAMES

```
#define CH_STATE_NAMES
```

Value:

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

Thread states as array of strings.

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

7.17.2.26 CH_FLAG_MODE_MASK

```
#define CH_FLAG_MODE_MASK (tmode_t)3U
```

Thread memory mode mask.

7.17.2.27 CH_FLAG_MODE_STATIC

```
#define CH_FLAG_MODE_STATIC (tmode_t)0U
```

Static thread.

7.17.2.28 CH_FLAG_MODE_HEAP

```
#define CH_FLAG_MODE_HEAP (tmode_t)1U
```

Thread allocated from a Memory Heap.

7.17.2.29 CH_FLAG_MODE_MPOOL

```
#define CH_FLAG_MODE_MPOOL (tmode_t) 2U
```

Thread allocated from a Memory Pool.

7.17.2.30 CH_FLAG_TERMINATE

```
#define CH_FLAG_TERMINATE (tmode_t) 4U
```

Termination requested flag.

7.17.2.31 firstprio

```
#define firstprio( rlp ) ((rlp)->next->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.

7.17.2.32 __sch_get_currthread

```
#define __sch_get_currthread( ) __instance_get_currthread(currcore)
```

Current thread pointer get macro.

Note

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

7.17.3 Function Documentation

7.17.3.1 __sch_ready_behind()

```
static thread_t* __sch_ready_behind (
    thread_t * tp ) [static]
```

Inserts a thread in the Ready List placing it behind its peers.

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 `next` and `prev` or list corruption would occur.

Postcondition

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

Parameters

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

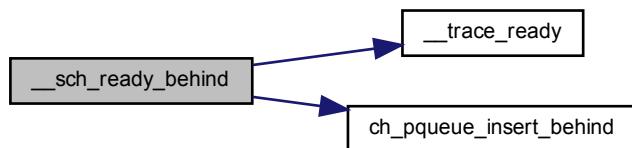
Returns

The thread pointer.

Function Class:

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

Here is the call graph for this function:

**7.17.3.2 __sch_ready_ahead()**

```
static thread_t* __sch_ready_ahead (
    thread_t * tp ) [static]
```

Inserts a thread in the Ready List placing it ahead its peers.

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

Precondition

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

Postcondition

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

Parameters

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

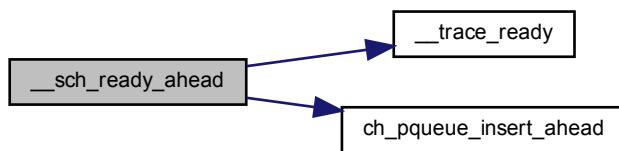
Returns

The thread pointer.

Function Class:

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

Here is the call graph for this function:

**7.17.3.3 __sch_reschedule_behind()**

```
static void __sch_reschedule_behind (
    void ) [static]
```

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.

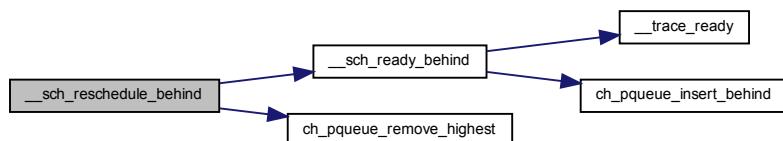
Note

Not a user function, it is meant to be invoked by the scheduler itself.

Function Class:

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

Here is the call graph for this function:



7.17.3.4 __sch_reschedule_ahead()

```
static void __sch_reschedule_ahead (
    void ) [static]
```

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.

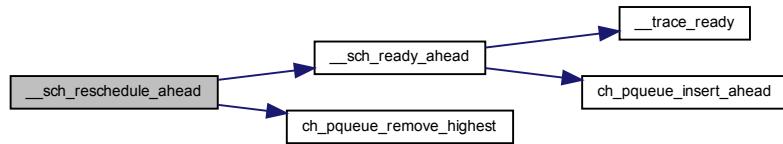
Note

Not a user function, it is meant to be invoked by the scheduler itself.

Function Class:

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

Here is the call graph for this function:



7.17.3.5 ch_sch_prio_insert()

```
static void ch_sch_prio_insert (
    ch_queue_t * qp,
    ch_queue_t * tp ) [inline]
```

Inserts a thread into a priority ordered queue.

Note

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

Parameters

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

Function Class:

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

7.17.3.6 chSchReadyI()

```
thread_t * chSchReadyI (
    thread_t * tp )
```

Inserts a thread in the Ready List placing it behind its peers.

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 `next` and `prev` or list corruption would occur.

Postcondition

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

Parameters

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

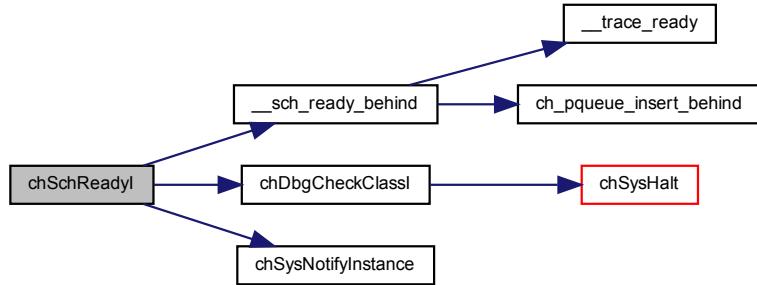
Returns

The thread pointer.

Function Class:

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

Here is the call graph for this function:

**7.17.3.7 chSchGoSleepS()**

```
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`.

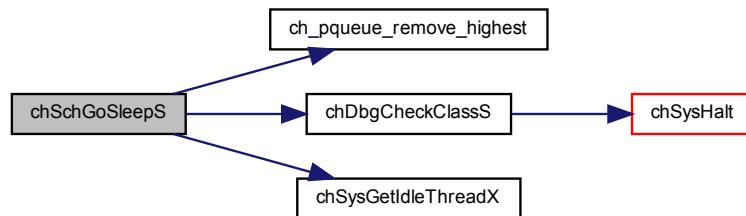
Parameters

in	<code>newstate</code>	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:



7.17.3.8 chSchGoSleepTimeoutS()

```
msg_t chSchGoSleepTimeoutS (
    tstate_t newstate,
    sysinterval_t timeout )
```

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

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

Parameters

in	<i>newstate</i>	the new thread state
in	<i>timeout</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> the thread enters an infinite sleep state, this is equivalent to invoking <code>chSchGoSleepS()</code> but, of course, less efficient. • <code>TIME_IMMEDIATE</code> this value is not allowed.

Returns

The wakeup message.

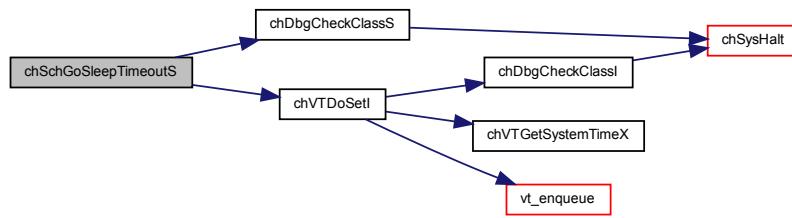
Return values

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

Function Class:

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

Here is the call graph for this function:

**7.17.3.9 chSchWakeupS()**

```
void chSchWakeupS (
    thread_t * ntp,
    msg_t msg )
```

Wakes up a thread.

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

Precondition

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

Note

It is equivalent to a `chSchReadyI()` followed by a `chSchRescheduleS()` but much more efficient.

The function assumes that the current thread has the highest priority.

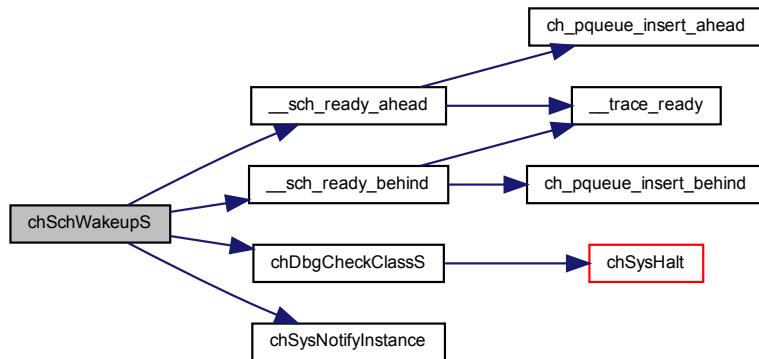
Parameters

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

Function Class:

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

Here is the call graph for this function:

**7.17.3.10 chSchRescheduleS()**

```
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.

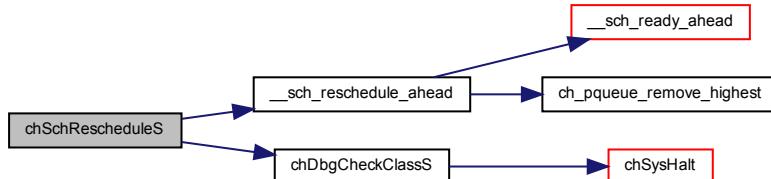
Note

Only local threads are considered, other cores are signaled and perform a reschedule locally.

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:



7.17.3.11 chSchIsPreemptionRequired()

```
bool chSchIsPreemptionRequired (
    void )
```

Evaluates if preemption is required.

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

Note

Not a user function, it is meant to be invoked from within the port layer in the IRQ-related preemption code.

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.

7.17.3.12 chSchDoPreemption()

```
void chSchDoPreemption (
    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.

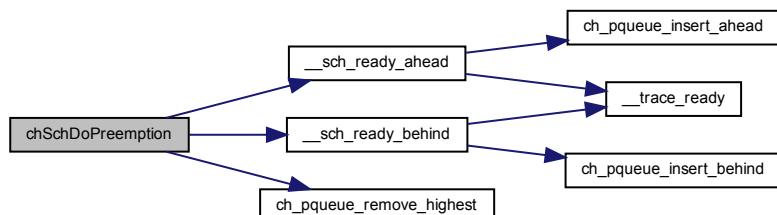
Note

Not a user function, it is meant to be invoked from within the port layer in the IRQ-related preemption code.

Function Class:

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

Here is the call graph for this function:



7.17.3.13 chSchPreemption()

```
void chSchPreemption (
    void )
```

All-in-one preemption code.

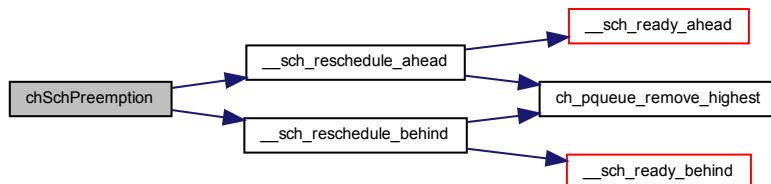
Note

Not a user function, it is meant to be invoked from within the port layer in the IRQ-related preemption code.

Function Class:

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

Here is the call graph for this function:



7.17.3.14 chSchDoYieldS()

```
void chSchDoYieldS (
    void )
```

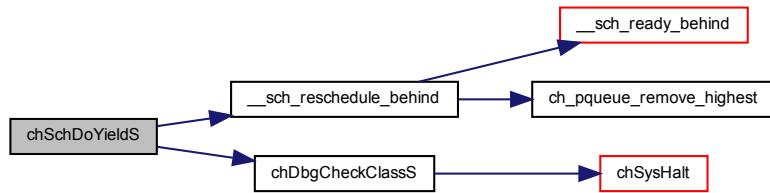
Yields the time slot.

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

Function Class:

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

Here is the call graph for this function:

**7.17.3.15 chSchSelectFirstI()**

```
thread_t * chSchSelectFirstI (
    void )
```

Makes runnable the fist thread in the ready list, does not reschedule internally.

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.

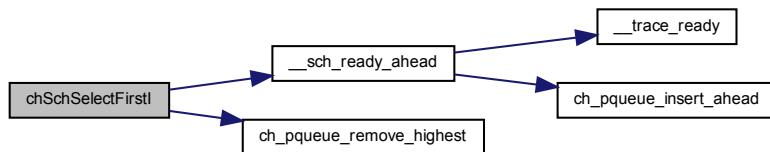
Returns

The pointer to the thread being switched in.

Function Class:

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

Here is the call graph for this function:



7.18 Base Kernel Services

7.18.1 Detailed Description

Modules

- [System Management](#)
- [Memory Alignment](#)
- [Time and Intervals](#)
- [Virtual Timers](#)
- [Threads](#)
- [Time Measurement](#)

7.19 System Management

7.19.1 Detailed Description

System related APIs and services:

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

Macros

- `#define CH_SYS_CORE0_MEMORY PORT_CORE0_BSS_SECTION`
Core zero memory affinity macro.
- `#define CH_SYS_CORE1_MEMORY PORT_CORE1_BSS_SECTION`
Core one memory affinity macro.
- `#define currcore ch_system.instances[port_get_core_id()]`
Access to current core's instance structure.
- `#define chSysGetRealtimeCounterX() (rtcnt_t)port_rt_get_counter_value()`
Returns the current value of the system real time counter.
- `#define chSysSwitch(ntp, otp)`
Performs a context switch.

Masks of executable integrity checks.

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

ISRs abstraction macros

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

Fast ISRs abstraction macros

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

Time conversion utilities for the realtime counter

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

Functions

- static `CH_SYS_CORE0_MEMORY THD_WORKING_AREA (ch_c0_idle_thread_wa, PORT_IDLE_THRESHOLD, AD_STACK_SIZE)`
Working area for core 0 idle thread.
- static `CH_SYS_CORE1_MEMORY THD_WORKING_AREA (ch_c1_idle_thread_wa, PORT_IDLE_THRESHOLD, AD_STACK_SIZE)`
Working area for core 1 idle thread.
- void `chSysWaitSystemState (system_state_t state)`
Waits for the system state to be equal to the specified one.
- void `chSysInit (void)`
System initialization.
- void `chSysHalt (const char *reason)`
Halts the system.
- bool `chSysIntegrityCheck (unsigned testmask)`
System integrity check.
- void `chSysTimerHandler (void)`
Handles time ticks for round robin preemption and timer increments.
- `syssts_t chSysGetStatusAndLockX (void)`
Returns the execution status and enters a critical zone.
- void `chSysRestoreStatusX (syssts_t sts)`
Restores the specified execution status and leaves a critical zone.
- bool `chSysIsCounterWithinX (rtcnt_t cnt, rtcnt_t start, rtcnt_t end)`
Realtime window test.
- void `chSysPolledDelayX (rtcnt_t cycles)`
Polled delay.
- static void `chSysDisable (void)`
Raises the system interrupt priority mask to the maximum level.
- static void `chSysSuspend (void)`
Raises the system interrupt priority mask to system level.

- static void [chSysEnable](#) (void)
Lowers the system interrupt priority mask to user level.
- static void [chSysLock](#) (void)
Enters the kernel lock state.
- static void [chSysUnlock](#) (void)
Leaves the kernel lock state.
- static void [chSysLockFromISR](#) (void)
Enters the kernel lock state from within an interrupt handler.
- static void [chSysUnlockFromISR](#) (void)
Leaves the kernel lock state from within an interrupt handler.
- static void [chSysUnconditionalLock](#) (void)
Unconditionally enters the kernel lock state.
- static void [chSysUnconditionalUnlock](#) (void)
Unconditionally leaves the kernel lock state.
- static void [chSysNotifyInstance](#) ([os_instance_t](#) *oip)
Notifies an OS instance to check for reschedule.
- static [thread_t](#) * [chSysGetIdleThreadX](#) (void)
Returns a pointer to the idle thread.

Variables

- [ch_system_t ch_system](#)
System root object.
- [CH_SYS_CORE0_MEMORY os_instance_t ch0](#)
Core 0 OS instance.
- const [os_instance_config_t ch_core0_cfg](#)
Core 0 OS instance configuration.
- [CH_SYS_CORE1_MEMORY os_instance_t ch1](#)
Core 1 OS instance.
- const [os_instance_config_t ch_core1_cfg](#)
Core 1 OS instance configuration.

7.19.2 Macro Definition Documentation

7.19.2.1 CH_SYS_CORE0_MEMORY

```
#define CH_SYS_CORE0_MEMORY PORT_CORE0_BSS_SECTION
```

Core zero memory affinity macro.

Note

The memory is meant to be reachable by both cores but preferred by core zero.
 Only uninitialized variables can be tagged with this attribute.

7.19.2.2 CH_SYS_CORE1_MEMORY

```
#define CH_SYS_CORE1_MEMORY PORT_CORE1_BSS_SECTION
```

Core one memory affinity macro.

Note

The memory is meant to be reachable by both cores but preferred by core one.
Only uninitialized variables can be tagged with this attribute.

7.19.2.3 currcore

```
#define currcore ch_system.instances[port_get_core_id()]
```

Access to current core's instance structure.

7.19.2.4 CH_IRQ_IS_VALID_PRIORITY

```
#define CH_IRQ_IS_VALID_PRIORITY(  
    prio ) PORT_IRQ_IS_VALID_PRIORITY(prio)
```

Priority level validation macro.

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

Parameters

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

Returns

Priority range result.

Return values

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

7.19.2.5 CH_IRQ_IS_VALID_KERNEL_PRIORITY

```
#define CH_IRQ_IS_VALID_KERNEL_PRIORITY(
```

```
prio ) PORT_IRQ_IS_VALID_KERNEL_PRIORITY (prio)
```

Priority level validation macro.

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

Parameters

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

Returns

Priority range result.

Return values

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

7.19.2.6 CH_IRQ_PROLOGUE

```
#define CH_IRQ_PROLOGUE( )
```

Value:

```
PORT_IRQ_PROLOGUE();  
CH_CFG_IRQ_PROLOGUE_HOOK();  
__stats_increase_irq();  
__trace_isr_enter(__func__);  
__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.

7.19.2.7 CH_IRQ_EPILOGUE

```
#define CH_IRQ_EPILOGUE( )
```

Value:

```
_dbg_check_leave_isr();  
_trace_isr_leave(__func__);  
CH_CFG_IRQ_EPILOGUE_HOOK();  
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.

7.19.2.8 CH_IRQ_HANDLER

```
#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.

7.19.2.9 CH_FAST_IRQ_HANDLER

```
#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.

7.19.2.10 S2RTC

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

Seconds to realtime counter.

Converts from seconds to realtime counter cycles.

Note

The macro assumes that `freq` ≥ 1 .

Parameters

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

Returns

The number of cycles.

Function Class:

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

7.19.2.11 MS2RTC

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

Milliseconds to realtime counter.

Converts from milliseconds to realtime counter cycles.

Note

The result is rounded upward to the next millisecond boundary.

The macro assumes that `freq` ≥ 1000 .

Parameters

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

Returns

The number of cycles.

Function Class:

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

7.19.2.12 US2RTC

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

Microseconds to realtime counter.

Converts from microseconds to realtime counter cycles.

Note

The result is rounded upward to the next microsecond boundary.

The macro assumes that `freq >= 1000000`.

Parameters

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

Returns

The number of cycles.

Function Class:

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

7.19.2.13 RTC2S

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

Realtime counter cycles to seconds.

Converts from realtime counter cycles number to seconds.

Note

The result is rounded up to the next second boundary.

The macro assumes that `freq >= 1`.

Parameters

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

Returns

The number of seconds.

Function Class:

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

7.19.2.14 RTC2MS

```
#define RTC2MS(
```

$$\quad \quad freq,$$

$$\quad \quad n) \(((n) - 1UL) / ((freq) / 1000UL)) + 1UL)$$

Realtime counter cycles to milliseconds.

Converts from realtime counter cycles number to milliseconds.

Note

The result is rounded up to the next millisecond boundary.

The macro assumes that *freq* ≥ 1000 .

Parameters

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

Returns

The number of milliseconds.

Function Class:

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

7.19.2.15 RTC2US

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

Realtime counter cycles to microseconds.

Converts from realtime counter cycles number to microseconds.

Note

The result is rounded up to the next microsecond boundary.

The macro assumes that `freq` ≥ 1000000 .

Parameters

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

Returns

The number of microseconds.

Function Class:

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

7.19.2.16 chSysGetRealtimeCounterX

```
#define chSysGetRealtimeCounterX( ) (rtcnt_t)port_rt_get_counter_value()
```

Returns the current value of the system real time counter.

Note

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

Returns

The value of the system realtime counter of type `rtcnt_t`.

Function Class:

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

7.19.2.17 chSysSwitch

```
#define chSysSwitch(
    ntp,
    otp
)
```

Value:

```
\

__trace_switch(ntp, otp);
__stats_ctxswc(ntp, otp);
CH_CFG_CONTEXT_SWITCH_HOOK(ntp, otp);
port_switch(ntp, otp);
}
```

Performs a context switch.

Note

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

Parameters

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

Function Class:

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

7.19.3 Function Documentation

7.19.3.1 THD_WORKING_AREA() [1/2]

```
static CH_SYS_CORE0_MEMORY THD_WORKING_AREA (
    ch_c0_idle_thread_wa ,
    PORT_IDLE_THREAD_STACK_SIZE ) [static]
```

Working area for core 0 idle thread.

7.19.3.2 THD_WORKING_AREA() [2/2]

```
static CH_SYS_CORE1_MEMORY THD_WORKING_AREA (
    ch_c1_idle_thread_wa ,
    PORT_IDLE_THREAD_STACK_SIZE ) [static]
```

Working area for core 1 idle thread.

7.19.3.3 chSysWaitSystemState()

```
void chSysWaitSystemState (
    system_state_t state )
```

Waits for the system state to be equal to the specified one.

Note

Can be called before `chSchObjectInit()` in order to wait for system initialization by another core.

Function Class:

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

7.19.3.4 chSysInit()

```
void chSysInit (
    void )
```

System initialization.

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

Precondition

Interrupts must disabled before invoking this function.

Postcondition

The main thread is created with priority NORMALPRIO and interrupts are enabled.
the system is in `ch_sys_running` state.

Function Class:

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

7.19.3.5 chSysHalt()

```
void chSysHalt (
    const char * reason )
```

Halts the system.

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

Note

Can be invoked from any system state.

Parameters

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

Function Class:

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

Here is the call graph for this function:

**7.19.3.6 chSysIntegrityCheckI()**

```
bool chSysIntegrityCheckI (
    unsigned testmask )
```

System integrity check.

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

Note

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

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

This function is not used internally, it is up to the application to define if and where to perform system checking.

Performing all tests at once can be a slow operation and can degrade the system response time. It is suggested to execute one test at time and release the critical zone in between tests.

Parameters

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

Returns

The test result.

Return values

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

Function Class:

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

Here is the call graph for this function:



7.19.3.7 chSysTimerHandlerI()

```
void chSysTimerHandlerI (
    void )
```

Handles time ticks for round robin preemption and timer increments.

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

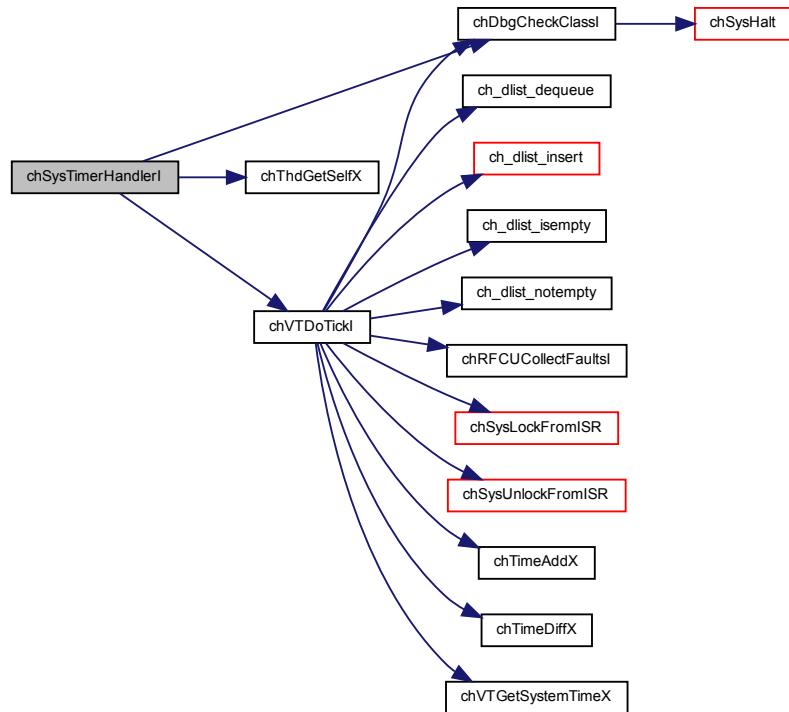
Note

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

Function Class:

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

Here is the call graph for this function:



7.19.3.8 chSysGetStatusAndLockX()

```
syssts_t chSysGetStatusAndLockX (
    void )
```

Returns the execution status and enters a critical zone.

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

Postcondition

The system is in a critical zone.

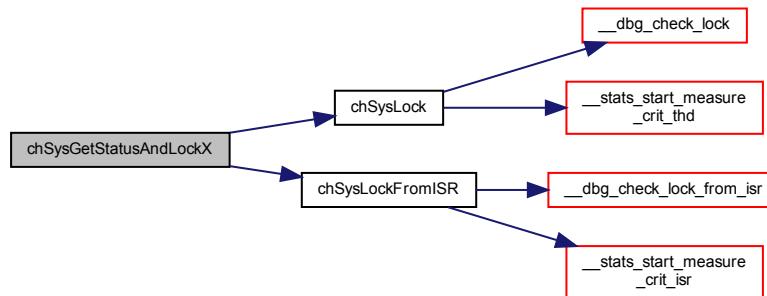
Returns

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

Function Class:

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

Here is the call graph for this function:

**7.19.3.9 chSysRestoreStatusX()**

```
void chSysRestoreStatusX (
    syssts_t sts )
```

Restores the specified execution status and leaves a critical zone.

Note

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

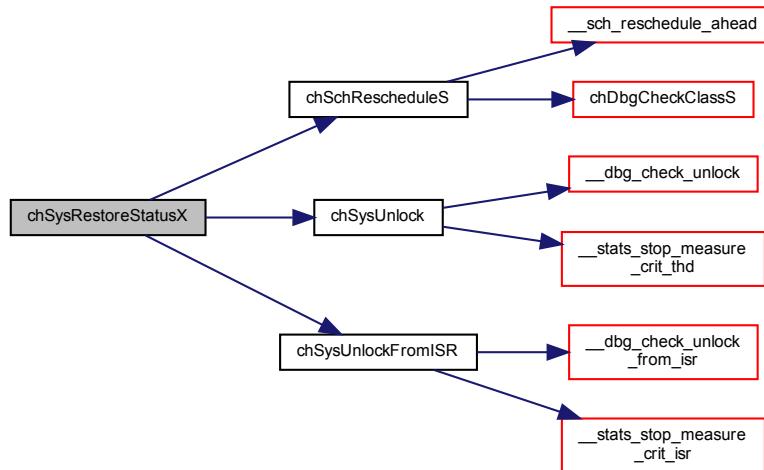
Parameters

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

Function Class:

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

Here is the call graph for this function:



7.19.3.10 chSysIsCounterWithinX()

```
bool chSysIsCounterWithinX (
    rtcnt_t cnt,
    rtcnt_t start,
    rtcnt_t end )
```

Realtime window test.

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

Note

When `start==end` then the function returns always false because a null time range is specified.

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

Parameters

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

Return values

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

Function Class:

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

7.19.3.11 chSysPolledDelayX()

```
void chSysPolledDelayX (
    rtcnt_t cycles )
```

Polled delay.

Note

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

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

Parameters

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

Function Class:

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

Here is the call graph for this function:

**7.19.3.12 chSysDisable()**

```
static void chSysDisable (
    void ) [inline], [static]
```

Raises the system interrupt priority mask to the maximum level.

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

Note

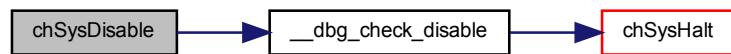
Do not invoke this API from within a kernel lock.

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

Function Class:

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

Here is the call graph for this function:



7.19.3.13 chSysSuspend()

```
static void chSysSuspend (
    void ) [inline], [static]
```

Raises the system interrupt priority mask to system level.

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

Note

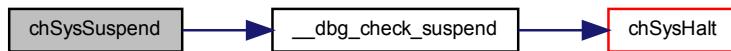
Do not invoke this API from within a kernel lock.

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

Function Class:

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

Here is the call graph for this function:



7.19.3.14 chSysEnable()

```
static void chSysEnable (
    void ) [inline], [static]
```

Lowers the system interrupt priority mask to user level.

All the interrupt sources are enabled.

Note

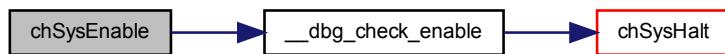
Do not invoke this API from within a kernel lock.

This API is no replacement for [chSysUnlock \(\)](#) which could do more than just enable interrupts.

Function Class:

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

Here is the call graph for this function:



7.19.3.15 chSysLock()

```
static void chSysLock (
    void ) [inline], [static]
```

Enters the kernel lock state.

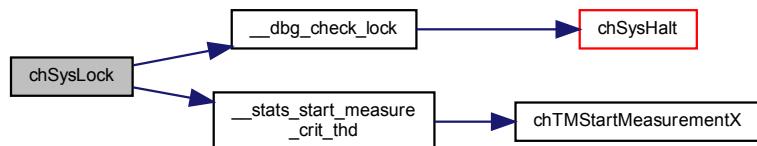
Note

The exact behavior of this function is port-dependent and could not be limited to disabling interrupts.

Function Class:

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

Here is the call graph for this function:



7.19.3.16 chSysUnlock()

```
static void chSysUnlock (
    void ) [inline], [static]
```

Leaves the kernel lock state.

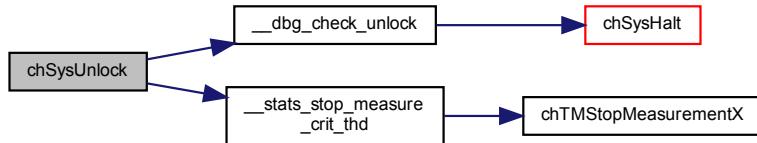
Note

The exact behavior of this function is port-dependent and could not be limited to enabling interrupts.

Function Class:

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

Here is the call graph for this function:



7.19.3.17 chSysLockFromISR()

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

Enters the kernel lock state from within an interrupt handler.

Note

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

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

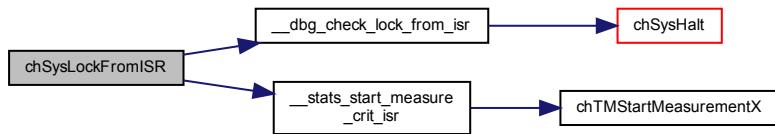
The exact behavior of this function is port-dependent and could not be limited to disabling interrupts.

This API must be invoked exclusively from interrupt handlers.

Function Class:

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

Here is the call graph for this function:

**7.19.3.18 chSysUnlockFromISR()**

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

Leaves the kernel lock state from within an interrupt handler.

Note

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

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

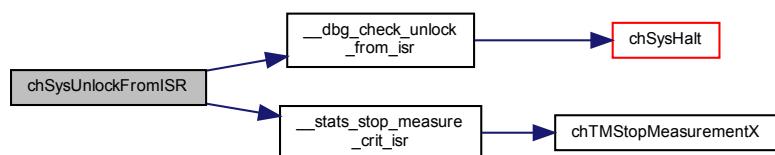
The exact behavior of this function is port-dependent and could not be limited to enabling interrupts.

This API must be invoked exclusively from interrupt handlers.

Function Class:

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

Here is the call graph for this function:



7.19.3.19 chSysUnconditionalLock()

```
static void chSysUnconditionalLock (
    void ) [inline], [static]
```

Unconditionally enters the kernel lock state.

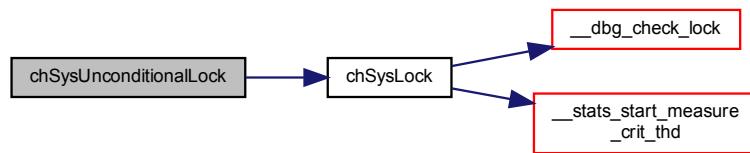
Note

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

Function Class:

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

Here is the call graph for this function:



7.19.3.20 chSysUnconditionalUnlock()

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

Unconditionally leaves the kernel lock state.

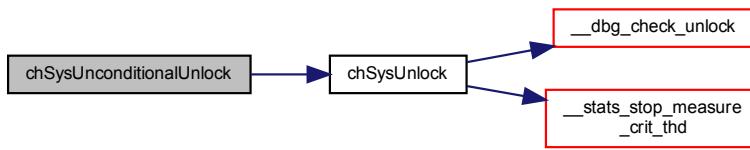
Note

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

Function Class:

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

Here is the call graph for this function:



7.19.3.21 chSysNotifyInstance()

```
static void chSysNotifyInstance (
    os_instance_t * oip ) [inline], [static]
```

Notifies an OS instance to check for reschedule.

An OS instance is notified to check if a reschedule is required, the implementation is port-dependent.

Parameters

in	<i>oip</i>	pointer to the instance to be notified
----	------------	--

7.19.3.22 chSysGetIdleThreadX()

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

Returns a pointer to the idle thread.

Precondition

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

Note

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

Returns

Pointer to the idle thread.

Function Class:

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

7.19.4 Variable Documentation

7.19.4.1 ch_system

`ch_system_t ch_system`

System root object.

7.19.4.2 ch0

```
CH_SYS_CORE0_MEMORY os_instance_t ch0
```

Core 0 OS instance.

7.19.4.3 ch_core0_cfg

```
const os_instance_config_t ch_core0_cfg
```

Initial value:

```
= {  
    .name          = "c0",  
    .mainthread_base = &__main_thread_stack_base__,  
    .mainthread_end   = &__main_thread_stack_end__,  
    .idlethread_base  = THD_WORKING_AREA_BASE(ch_c0_idle_thread_wa),  
    .idlethread_end    = THD_WORKING_AREA_END(ch_c0_idle_thread_wa)  
}
```

Core 0 OS instance configuration.

7.19.4.4 ch1

```
CH_SYS_CORE1_MEMORY os_instance_t ch1
```

Core 1 OS instance.

7.19.4.5 ch_core1_cfg

```
const os_instance_config_t ch_core1_cfg
```

Initial value:

```
= {  
    .name          = "c1",  
    .mainthread_base = &__c1_main_thread_stack_base__,  
    .mainthread_end   = &__c1_main_thread_stack_end__,  
    .idlethread_base  = THD_WORKING_AREA_BASE(ch_c1_idle_thread_wa),  
    .idlethread_end    = THD_WORKING_AREA_END(ch_c1_idle_thread_wa)  
}
```

Core 1 OS instance configuration.

7.20 Memory Alignment

7.20.1 Detailed Description

Memory Alignment services.

Memory alignment support macros

- `#define MEM_ALIGN_MASK(a) ((size_t)(a) - 1U)`
Alignment mask constant.
- `#define MEM_ALIGN_PREV(p, a)`
Aligns to the previous aligned memory address.
- `#define MEM_ALIGN_NEXT(p, a)`
Aligns to the next aligned memory address.
- `#define MEM_IS_ALIGNED(p, a) (((size_t)(p) & MEM_ALIGN_MASK(a)) == 0U)`
Returns whatever a pointer or memory size is aligned.
- `#define MEM_IS_VALID_ALIGNMENT(a) (((size_t)(a) != 0U) && (((size_t)(a) & ((size_t)(a) - 1U)) == 0U))`
Returns whatever a constant is a valid alignment.

7.20.2 Macro Definition Documentation

7.20.2.1 MEM_ALIGN_MASK

```
#define MEM_ALIGN_MASK(
    a ) ((size_t)(a) - 1U)
```

Alignment mask constant.

Parameters

in	a	alignment, must be a power of two
----	---	-----------------------------------

7.20.2.2 MEM_ALIGN_PREV

```
#define MEM_ALIGN_PREV(
    p,
    a )
```

Value:

```
/*lint -save -e9033 [10.8] The cast is safe.*/
((size_t)(p) & ~MEM_ALIGN_MASK(a))
/*lint -restore*/\
```

Aligns to the previous aligned memory address.

Parameters

in	<i>p</i>	variable to be aligned
in	<i>a</i>	alignment, must be a power of two

7.20.2.3 MEM_ALIGN_NEXT

```
#define MEM_ALIGN_NEXT (
    p,
    a )
```

Value:

```
/*lint -save -e9033 [10.8] The cast is safe.*/
MEM_ALIGN_PREV((size_t)(p) + MEM_ALIGN_MASK(a), (a)) \
/*lint -restore*/
```

Aligns to the next aligned memory address.

Parameters

in	<i>p</i>	variable to be aligned
in	<i>a</i>	alignment, must be a power of two

7.20.2.4 MEM_IS_ALIGNED

```
#define MEM_IS_ALIGNED (
    p,
    a ) (((size_t)(p) & MEM_ALIGN_MASK(a)) == 0U)
```

Returns whatever a pointer or memory size is aligned.

Parameters

in	<i>p</i>	variable to be aligned
in	<i>a</i>	alignment, must be a power of two

7.20.2.5 MEM_IS_VALID_ALIGNMENT

```
#define MEM_IS_VALID_ALIGNMENT (
    a ) (((size_t)(a) != 0U) && (((size_t)(a) & ((size_t)(a) - 1U)) == 0U))
```

Returns whatever a constant is a valid alignment.

Valid alignments are powers of two.

Parameters

in	<i>a</i>	alignment to be checked, must be a constant
----	----------	---

7.21 Time and Intervals

7.21.1 Detailed Description

This module is responsible for handling of system time and time intervals.

Special time constants

- `#define TIME_IMMEDIATE ((sysinterval_t)0)`
Zero interval specification for some functions with a timeout specification.
- `#define TIME_INFINITE ((sysinterval_t)-1)`
Infinite interval specification for all functions with a timeout specification.
- `#define TIME_MAX_INTERVAL ((sysinterval_t)-2)`
Maximum interval constant usable as timeout.
- `#define TIME_MAX_SYSTIME ((systime_t)-1)`
Maximum system of system time before it wraps.

Fast time conversion utilities

- `#define TIME_S2I(secs) ((sysinterval_t)((time_conv_t)(secs) * (time_conv_t)CH_CFG_ST_FREQUENCY))`
Seconds to time interval.
- `#define TIME_MS2I(msecs)`
Milliseconds to time interval.
- `#define TIME_US2I(usecs)`
Microseconds to time interval.
- `#define TIME_I2S(interval)`
Time interval to seconds.
- `#define TIME_I2MS(interval)`
Time interval to milliseconds.
- `#define TIME_I2US(interval)`
Time interval to microseconds.

Secure time conversion utilities

- `static sysinterval_t chTimeS2I (time_secs_t secs)`
Seconds to time interval.
- `static sysinterval_t chTimeMS2I (time_msecs_t msec)`
Milliseconds to time interval.
- `static sysinterval_t chTimeUS2I (time_usecs_t usec)`
Microseconds to time interval.
- `static time_secs_t chTimeI2S (sysinterval_t interval)`
Time interval to seconds.
- `static time_msecs_t chTimeI2MS (sysinterval_t interval)`
Time interval to milliseconds.
- `static time_usecs_t chTimeI2US (sysinterval_t interval)`
Time interval to microseconds.
- `static systime_t chTimeAddX (systime_t systime, sysinterval_t interval)`
Adds an interval to a system time returning a system time.

- static `sysinterval_t chTimeDiffX (systime_t start, systime_t end)`
Subtracts two system times returning an interval.
- static bool `chTimelsInRangeX (systime_t time, systime_t start, systime_t end)`
Checks if the specified time is within the specified time range.
- static `systimestamp_t chTimeStampAddX (systimestamp_t stamp, sysinterval_t interval)`
Adds an interval to a time stamp returning a time stamp.
- static `sysinterval_t chTimeStampDiffX (systimestamp_t start, systimestamp_t end)`
Subtracts two time stamps returning an interval.
- static bool `chTimeStampsInRangeX (systimestamp_t stamp, systimestamp_t start, systimestamp_t end)`
Checks if the specified time stamp is within the specified time stamps range.

Typedefs

- `typedef uint64_t systime_t`
Type of system time.
- `typedef uint64_t sysinterval_t`
Type of time interval.
- `typedef uint64_t systimestamp_t`
Type of a time stamp.
- `typedef uint32_t time_secs_t`
Type of seconds.
- `typedef uint32_t time_msecs_t`
Type of milliseconds.
- `typedef uint32_t time_usecs_t`
Type of microseconds.
- `typedef uint64_t time_conv_t`
Type of time conversion variable.

7.21.2 Macro Definition Documentation

7.21.2.1 TIME_IMMEDIATE

```
#define TIME_IMMEDIATE ((sysinterval_t)0)
```

Zero interval specification for some functions with a timeout specification.

Note

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

7.21.2.2 TIME_INFINITE

```
#define TIME_INFINITE ((sysinterval_t)-1)
```

Infinite interval specification for all functions with a timeout specification.

Note

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

7.21.2.3 TIME_MAX_INTERVAL

```
#define TIME_MAX_INTERVAL ((sysinterval_t)-2)
```

Maximum interval constant usable as timeout.

7.21.2.4 TIME_MAX_SYSTIME

```
#define TIME_MAX_SYSTIME ((systime_t)-1)
```

Maximum system of system time before it wraps.

7.21.2.5 TIME_S2I

```
#define TIME_S2I( secs ) ((sysinterval_t)((time_conv_t)(secs) * (time_conv_t)CH_CFG_ST_FREQUENCY))
```

Seconds to time interval.

Converts from seconds to system ticks number.

Note

The result is rounded upward to the next tick boundary.

Use of this macro for large values is not secure because integer overflows, make sure your value can be correctly converted.

Parameters

in	secs	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.

7.21.2.6 TIME_MS2I

```
#define TIME_MS2I(  
    msecs )
```

Value:

```
((sysinterval_t) (((time_conv_t)(msecs) *  
    (time_conv_t)CH_CFG_ST_FREQUENCY) +  
    (time_conv_t)999) / (time_conv_t)1000)) \\\
```

Milliseconds to time interval.

Converts from milliseconds to system ticks number.

Note

The result is rounded upward to the next tick boundary.

Use of this macro for large values is not secure because integer overflows, make sure your value can be correctly converted.

Parameters

in	msecs	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.

7.21.2.7 TIME_US2I

```
#define TIME_US2I(  
    usecs )
```

Value:

```
((sysinterval_t) (((time_conv_t)(usecs) *  
    (time_conv_t)CH_CFG_ST_FREQUENCY) +  
    (time_conv_t)999999) / (time_conv_t)1000000)) \\\
```

Microseconds to time interval.

Converts from microseconds to system ticks number.

Note

The result is rounded upward to the next tick boundary.

Use of this macro for large values is not secure because integer overflows, make sure your value can be correctly converted.

Parameters

in	<i>usecs</i>	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.

7.21.2.8 TIME_I2S

```
#define TIME_I2S(  
    interval )
```

Value:

```
(time_secs_t) (((time_conv_t)(interval) +  
    (time_conv_t)CH_CFG_ST_FREQUENCY -  
    (time_conv_t)1) / (time_conv_t)CH_CFG_ST_FREQUENCY)
```

Time interval to seconds.

Converts from system ticks number to seconds.

Note

The result is rounded up to the next second boundary.

Use of this macro for large values is not secure because integer overflows, make sure your value can be correctly converted.

Parameters

in	<i>interval</i>	interval in 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.

7.21.2.9 TIME_I2MS

```
#define TIME_I2MS(
    interval )
```

Value:

$$\text{time_msecs_t} (((\text{time_conv_t})(\text{interval}) * (\text{time_conv_t})1000) + \\ (\text{time_conv_t})\text{CH_CFG_ST_FREQUENCY} - (\text{time_conv_t})1) / \\ (\text{time_conv_t})\text{CH_CFG_ST_FREQUENCY})$$

Time interval to milliseconds.

Converts from system ticks number to milliseconds.

Note

The result is rounded up to the next millisecond boundary.

Use of this macro for large values is not secure because integer overflows, make sure your value can be correctly converted.

Parameters

in	interval	interval in 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.

7.21.2.10 TIME_I2US

```
#define TIME_I2US(
    interval )
```

Value:

$$\text{time_msecs_t} (((\text{time_conv_t})(\text{interval}) * (\text{time_conv_t})1000000) + \\ (\text{time_conv_t})\text{CH_CFG_ST_FREQUENCY} - (\text{time_conv_t})1) / \\ (\text{time_conv_t})\text{CH_CFG_ST_FREQUENCY})$$

Time interval to microseconds.

Converts from system ticks number to microseconds.

Note

The result is rounded up to the next microsecond boundary.

Use of this macro for large values is not secure because integer overflows, make sure your value can be correctly converted.

Parameters

in	<i>interval</i>	interval in 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.

7.21.3 Typedef Documentation

7.21.3.1 `systime_t`

```
typedef uint64_t systime_t
```

Type of system time.

Note

It is selectable in configuration between 16, 32 or 64 bits.

7.21.3.2 `sysinterval_t`

```
typedef uint64_t sysinterval_t
```

Type of time interval.

Note

It is selectable in configuration between 16, 32 or 64 bits.

7.21.3.3 `systimestamp_t`

```
typedef uint64_t systimestamp_t
```

Type of a time stamp.

7.21.3.4 time_secs_t

```
typedef uint32_t time_secs_t
```

Type of seconds.

Note

It is selectable in configuration between 16 or 32 bits.

7.21.3.5 time_msecs_t

```
typedef uint32_t time_msecs_t
```

Type of milliseconds.

Note

It is selectable in configuration between 16 or 32 bits.

7.21.3.6 time_usecs_t

```
typedef uint32_t time_usecs_t
```

Type of microseconds.

Note

It is selectable in configuration between 16 or 32 bits.

7.21.3.7 time_conv_t

```
typedef uint64_t time_conv_t
```

Type of time conversion variable.

Note

This type must have double width than other time types, it is only used internally for conversions.

7.21.4 Function Documentation

7.21.4.1 chTimeS2I()

```
static sysinterval_t chTimeS2I (
    time_secs_t secs ) [inline], [static]
```

Seconds to time interval.

Converts from seconds to system ticks number.

Note

The result is rounded upward to the next tick boundary.

Parameters

in	secs	number of seconds
----	------	-------------------

Returns

The number of ticks.

Function Class:

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

7.21.4.2 chTimeMS2I()

```
static sysinterval_t chTimeMS2I (
    time_msecs_t msec ) [inline], [static]
```

Milliseconds to time interval.

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:

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

7.21.4.3 chTimeUS2I()

```
static sysinterval_t chTimeUS2I (
    time_usecs_t usec ) [inline], [static]
```

Microseconds to time interval.

Converts from microseconds to system ticks number.

Note

The result is rounded upward to the next tick boundary.

Parameters

in	<i>usec</i>	number of microseconds
----	-------------	------------------------

Returns

The number of ticks.

Function Class:

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

7.21.4.4 chTimeI2S()

```
static time_secs_t chTimeI2S (
    sysinterval_t interval ) [inline], [static]
```

Time interval to seconds.

Converts from system interval to seconds.

Note

The result is rounded up to the next second boundary.

Parameters

in	<i>interval</i>	interval in ticks
----	-----------------	-------------------

Returns

The number of seconds.

Function Class:

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

7.21.4.5 chTimeI2MS()

```
static time_msecs_t chTimeI2MS (
    sysinterval_t interval ) [inline], [static]
```

Time interval to milliseconds.

Converts from system interval to milliseconds.

Note

The result is rounded up to the next millisecond boundary.

Parameters

in	<i>interval</i>	interval in ticks
----	-----------------	-------------------

Returns

The number of milliseconds.

Function Class:

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

7.21.4.6 chTimeI2US()

```
static time_usecs_t chTimeI2US (
    sysinterval_t interval ) [inline], [static]
```

Time interval to microseconds.

Converts from system interval to microseconds.

Note

The result is rounded up to the next microsecond boundary.

Parameters

in	<i>interval</i>	interval in ticks
----	-----------------	-------------------

Returns

The number of microseconds.

Function Class:

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

7.21.4.7 chTimeAddX()

```
static systime_t chTimeAddX (
    systime_t systime,
    sysinterval_t interval ) [inline], [static]
```

Adds an interval to a system time returning a system time.

Parameters

in	<i>systime</i>	base system time
in	<i>interval</i>	interval to be added

Returns

The new system time.

Function Class:

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

7.21.4.8 chTimeDiffX()

```
static sysinterval_t chTimeDiffX (
    systime_t start,
    systime_t end ) [inline], [static]
```

Subtracts two system times returning an interval.

Parameters

in	<i>start</i>	first system time
in	<i>end</i>	second system time

Returns

The interval representing the time difference.

Function Class:

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

7.21.4.9 chTimelsInRangeX()

```
static bool chTimeIsInRangeX (
    systime_t time,
    systime_t start,
    systime_t end ) [inline], [static]
```

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

Note

When *start*==*end* then the function returns always false because the time window has zero size.

Parameters

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

Return values

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

Function Class:

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

7.21.4.10 chTimeStampAddX()

```
static systimestamp_t chTimeStampAddX (
    systimestamp_t stamp,
    sysinterval_t interval ) [inline], [static]
```

Adds an interval to a time stamp returning a time stamp.

Parameters

in	<i>stamp</i>	base time stamp
in	<i>interval</i>	interval to be added

Returns

The new time stamp.

Function Class:

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

7.21.4.11 chTimeStampDiffX()

```
static sysinterval_t chTimeStampDiffX (
    systimestamp_t start,
    systimestamp_t end ) [inline], [static]
```

Subtracts two time stamps returning an interval.

Note

Intervals can then be used for converting in absolute time.

Parameters

in	<i>start</i>	first time stamp
in	<i>end</i>	second time stamp

Returns

The interval representing the time stamps difference.

Function Class:

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

7.21.4.12 chTimeStampIsInRangeX()

```
static bool chTimeStampIsInRangeX (
    systimestamp_t stamp,
    systimestamp_t start,
    systimestamp_t end ) [inline], [static]
```

Checks if the specified time stamp is within the specified time stamps range.

Note

When *start*==*end* then the function returns always false because the time window has zero size.

Parameters

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

Return values

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

Function Class:

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

7.22 Virtual Timers

7.22.1 Detailed Description

Time and Virtual Timers related APIs and services.

Functions

- static void `vt_insert_first` (`virtual_timers_list_t` *`vtlp`, `virtual_timer_t` *`vtp`, `systime_t` `now`, `sysinterval_t` `delay`)
Inserts a timer as first element in a delta list.
- static void `vt_enqueue` (`virtual_timers_list_t` *`vtlp`, `virtual_timer_t` *`vtp`, `systime_t` `now`, `sysinterval_t` `delay`)
Enqueues a virtual timer in a virtual timers list.
- void `chVTDoSetl` (`virtual_timer_t` *`vtp`, `sysinterval_t` `delay`, `vfunc_t` `vtfunc`, `void` *`par`)
Enables a one-shot virtual timer.
- void `chVTDoSetContinuousl` (`virtual_timer_t` *`vtp`, `sysinterval_t` `delay`, `vfunc_t` `vtfunc`, `void` *`par`)
Enables a continuous virtual timer.
- void `chVTDoResetl` (`virtual_timer_t` *`vtp`)
Disables a Virtual Timer.
- `sysinterval_t chVTGetRemainingIntervall` (`virtual_timer_t` *`vtp`)
Returns the remaining time interval before next timer trigger.
- void `chVTDotickl` (`void`)
Virtual timers ticker.
- `sysimestamp_t chVTGetTimeStampl` (`void`)
Generates a monotonic time stamp.
- void `chVTResetTimeStampl` (`void`)
Resets and re-synchronizes the time stamps monotonic counter.
- static void `chVTOBJECTInit` (`virtual_timer_t` *`vtp`)
Initializes a virtual_timer_t object.
- static `systime_t chVTGetSystemTimeX` (`void`)
Current system time.
- static `systime_t chVTGetSystemTime` (`void`)
Current system time.
- static `sysinterval_t chVTTIMEElapsedSinceX` (`systime_t` `start`)
Returns the elapsed time since the specified start time.
- static bool `chVTIsSystemTimeWithinX` (`systime_t` `start`, `systime_t` `end`)
Checks if the current system time is within the specified time window.
- static bool `chVTIsSystemTimeWithin` (`systime_t` `start`, `systime_t` `end`)
Checks if the current system time is within the specified time window.
- static bool `chVTGetTimersStatel` (`sysinterval_t` *`timep`)
Returns the time interval until the next timer event.
- static bool `chVTIsArmedl` (`const virtual_timer_t` *`vtp`)
Returns true if the specified timer is armed.
- static bool `chVTIsArmed` (`const virtual_timer_t` *`vtp`)
Returns true if the specified timer is armed.
- static void `chVTResetl` (`virtual_timer_t` *`vtp`)
Disables a Virtual Timer.
- static void `chVTReset` (`virtual_timer_t` *`vtp`)
Disables a Virtual Timer.
- static void `chVTSetl` (`virtual_timer_t` *`vtp`, `sysinterval_t` `delay`, `vfunc_t` `vtfunc`, `void` *`par`)

- static void `chVTSet (virtual_timer_t *vtp, sysinterval_t delay, vfunc_t vfunc, void *par)`

Enables a one-shot virtual timer.
- static void `chVTSetContinuousI (virtual_timer_t *vtp, sysinterval_t delay, vfunc_t vfunc, void *par)`

Enables a one-shot virtual timer.
- static void `chVTSetContinuous (virtual_timer_t *vtp, sysinterval_t delay, vfunc_t vfunc, void *par)`

Enables a continuous virtual timer.
- static `sysinterval_t chVTGetReloadIntervalX (virtual_timer_t *vtp)`

Returns the current reload value.
- static void `chVTSetReloadIntervalX (virtual_timer_t *vtp, sysinterval_t reload)`

Changes a timer reload time interval.
- static `systimestamp_t chVTGetTimeStamp (void)`

Generates a monotonic time stamp.
- static void `chVTRestTimeStamp (void)`

Resets and re-synchronizes the time stamps monotonic counter.
- static void `__vt_object_init (virtual_timers_list_t *vtlp)`

Virtual Timers instance initialization.

7.22.2 Function Documentation

7.22.2.1 vt_insert_first()

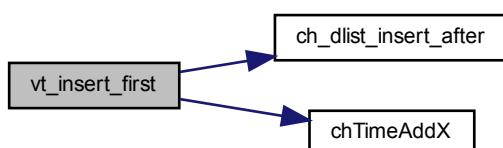
```
static void vt_insert_first (
    virtual_timers_list_t * vtlp,
    virtual_timer_t * vtp,
    systime_t now,
    sysinterval_t delay ) [static]
```

Inserts a timer as first element in a delta list.

Note

This is the special case when the delta list is initially empty.

Here is the call graph for this function:

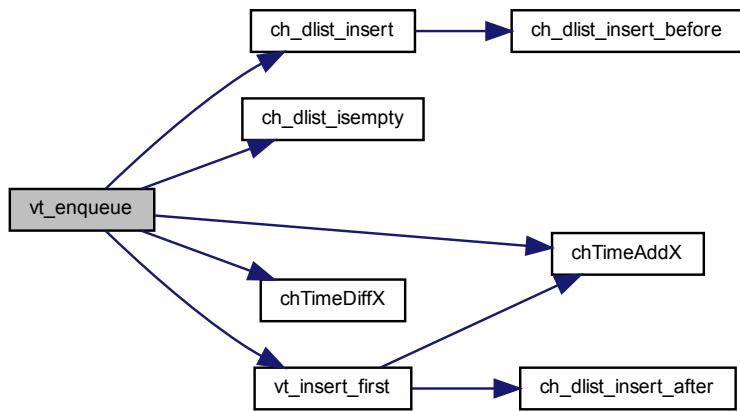


7.22.2.2 vt_enqueue()

```
static void vt_enqueue (
    virtual_timers_list_t * vtlp,
    virtual_timer_t * vtp,
    systime_t now,
    sysinterval_t delay ) [static]
```

Enqueues a virtual timer in a virtual timers list.

Here is the call graph for this function:



7.22.2.3 chVTDoSetI()

```
void chVTDoSetI (
    virtual_timer_t * vtp,
    sysinterval_t delay,
    vtfunc_t vtfunc,
    void * par )
```

Enables a one-shot virtual timer.

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

Precondition

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

Note

The callback function is invoked from interrupt context.

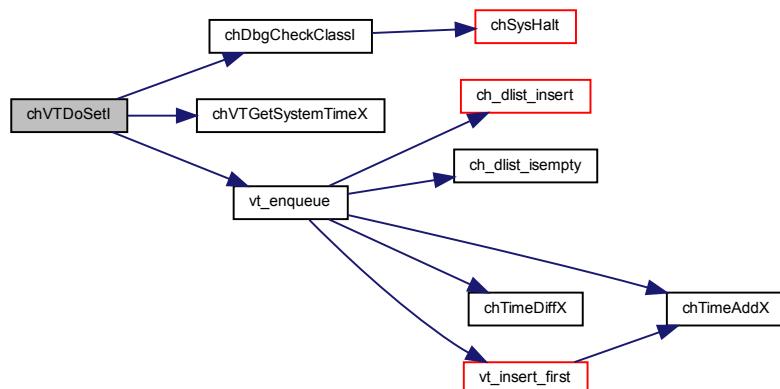
Parameters

out	<i>vtp</i>	the <code>virtual_timer_t</code> structure pointer
in	<i>delay</i>	the number of ticks before the operation timeouts, the special values are handled as follow: <ul style="list-style-type: none"> <code>TIME_INFINITE</code> is allowed but interpreted as a normal time specification. <code>TIME_IMMEDIATE</code> this value is not allowed.
in	<i>vfunc</i>	the timer callback function. After invoking the callback the timer is disabled and the structure can be disposed or reused.
in	<i>par</i>	a parameter that will be passed to the callback function

Function Class:

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

Here is the call graph for this function:



7.22.2.4 `chVTDoSetContinuousI()`

```

void chVTDoSetContinuousI (
    virtual_timer_t * vtp,
    sysinterval_t delay,
    vfunc_t vfunc,
    void * par )
  
```

Enables a continuous virtual timer.

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

Precondition

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

Note

The callback function is invoked from interrupt context.

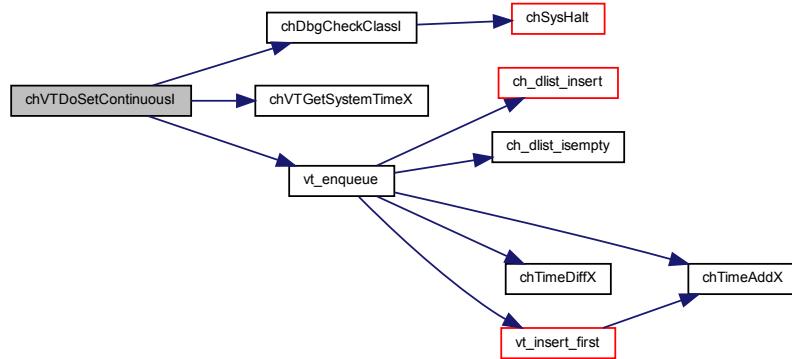
Parameters

out	<i>vtp</i>	the <code>virtual_timer_t</code> structure pointer
in	<i>delay</i>	the number of ticks before the operation timeouts, the special values are handled as follow: <ul style="list-style-type: none"> <code>TIME_INFINITE</code> is allowed but interpreted as a normal time specification. <code>TIME_IMMEDIATE</code> this value is not allowed.
in	<i>vfunc</i>	the timer callback function. After invoking the callback the timer is disabled and the structure can be disposed or reused.
in	<i>par</i>	a parameter that will be passed to the callback function

Function Class:

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

Here is the call graph for this function:



7.22.2.5 chVTDoResetI()

```
void chVTDoResetI (
    virtual_timer_t * vtp )
```

Disables a Virtual Timer.

Precondition

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

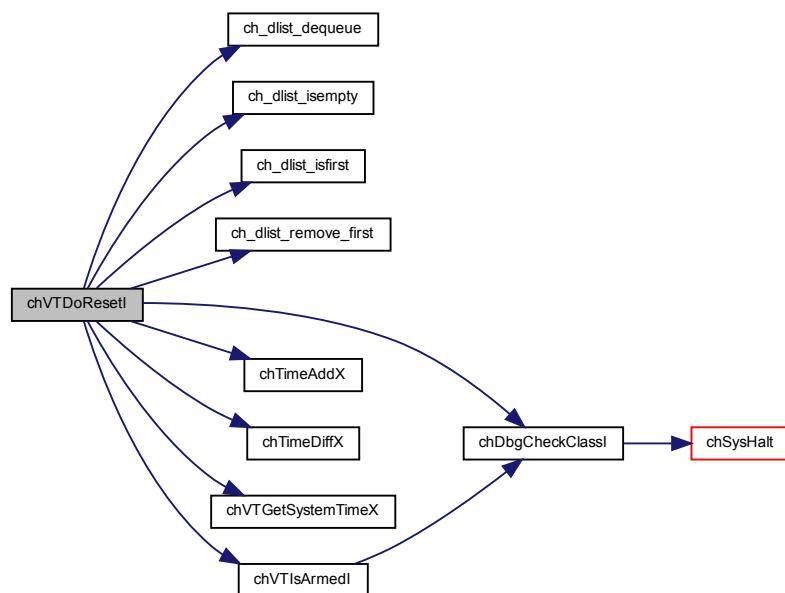
Parameters

in	<i>vtp</i>	the virtual_timer_t 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:

**7.22.2.6 chVTGetRemainingInterval()**

```
sysinterval_t chVTGetRemainingIntervalI (
    virtual_timer_t * vtp )
```

Returns the remaining time interval before next timer trigger.

Note

This function can be called while the timer is active.

Parameters

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

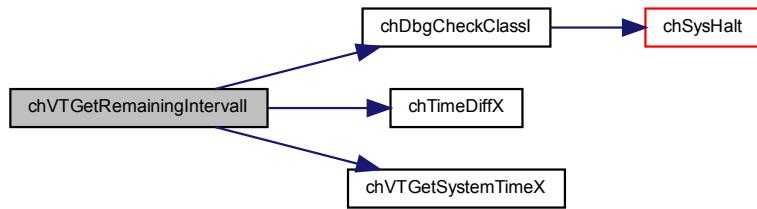
Returns

The remaining time 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:

**7.22.2.7 chVTDotickI()**

```
void chVTDotickI (
    void )
```

Virtual timers ticker.

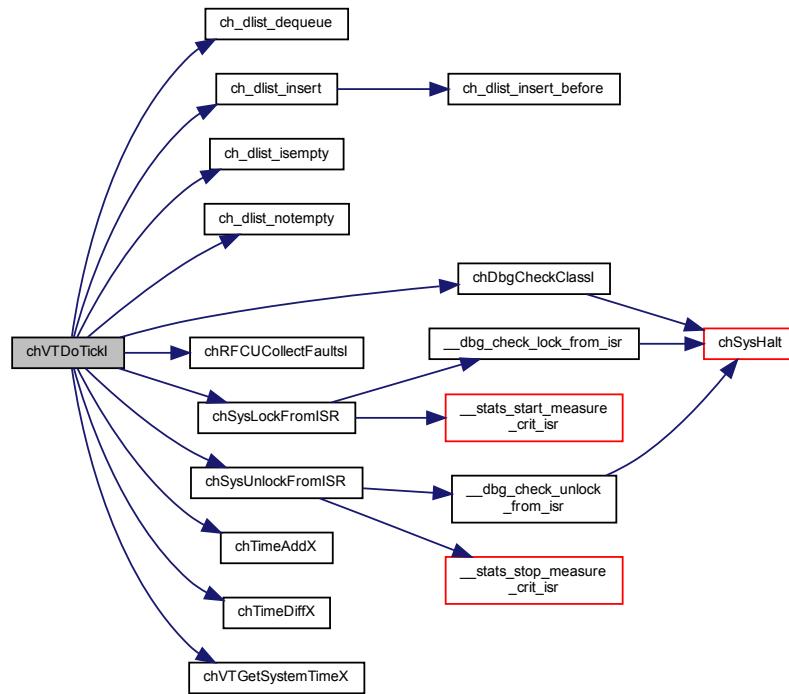
Note

The system lock is released before entering the callback and re-acquired immediately after. It is callback's responsibility to acquire the lock if needed. This is done in order to reduce interrupts jitter when many timers are in use.

Function Class:

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

Here is the call graph for this function:



7.22.2.8 chVTGetTimeStampI()

```
systimestamp_t chVTGetTimeStampI (
    void )
```

Generates a monotonic time stamp.

This function generates a monotonic time stamp synchronized with the system time. The time stamp has the same resolution of system time.

Note

There is an assumption, this function must be called at least once before the system time wraps back to zero or synchronization is lost. You may use a periodic virtual timer with a very large interval in order to keep time stamps synchronized by calling this function.

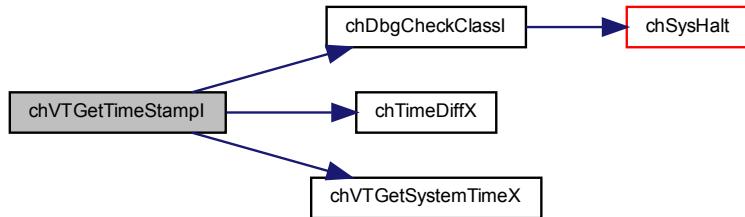
Returns

The time stamp.

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:

**7.22.2.9 chVTRestartTimeStampI()**

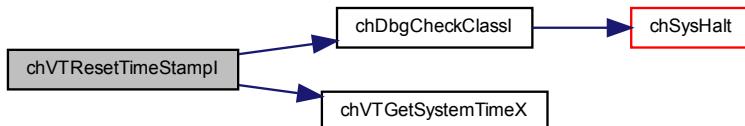
```
void chVTRestartTimeStampI (
    void )
```

Resets and re-synchronizes the time stamps monotonic counter.

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:



7.22.2.10 chVTOBJECTINIT()

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

Initializes a `virtual_timer_t` object.

Note

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

Parameters

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

Function Class:

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

7.22.2.11 chVTGetSystemTimeX()

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

Current system time.

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

Note

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

This function can be called from any context but its atomicity is not guaranteed on architectures whose word size is less than `systime_t` size.

Returns

The system time in ticks.

Function Class:

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

7.22.2.12 chVTGetSystemTime()

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

Current system time.

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

Note

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

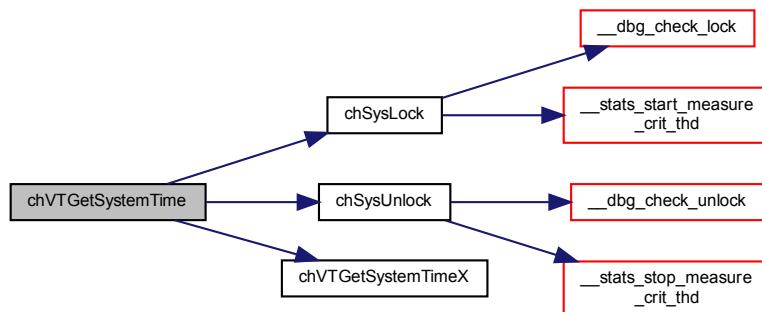
Returns

The system time in ticks.

Function Class:

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

Here is the call graph for this function:



7.22.2.13 chVTTimeElapsedSinceX()

```
static sysinterval_t chVTTimeElapsedSinceX (
    systime_t start ) [inline], [static]
```

Returns the elapsed time since the specified start time.

Parameters

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

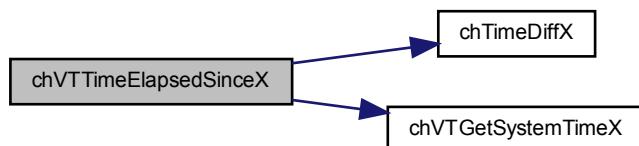
Returns

The elapsed time.

Function Class:

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

Here is the call graph for this function:

**7.22.2.14 chVTIsSystemTimeWithinX()**

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

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

Note

When `start==end` then the function returns always false because the time window has zero size.

Parameters

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

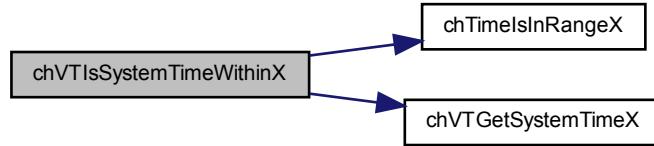
Return values

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

Function Class:

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

Here is the call graph for this function:



7.22.2.15 chVTIsSystemTimeWithin()

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

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

Note

When `start==end` then the function returns always false because the time window has zero size.

Parameters

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

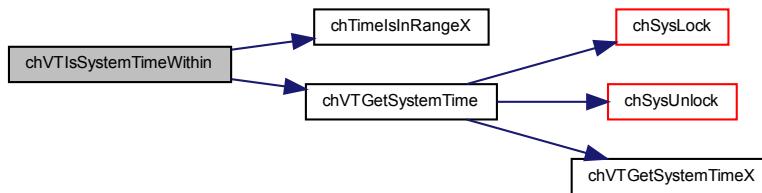
Return values

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

Function Class:

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

Here is the call graph for this function:

**7.22.2.16 chVTGetTimersStateI()**

```
static bool chVTGetTimersStateI (
    sysinterval_t * timep ) [inline], [static]
```

Returns the time interval until the next timer event.

Note

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

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

Parameters

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

Returns

The time, in ticks, until next time event.

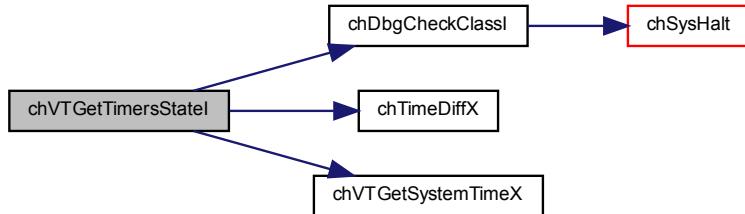
Return values

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

Function Class:

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

Here is the call graph for this function:

**7.22.2.17 chVTIsArmedI()**

```
static bool chVTIsArmedI (
    const virtual_timer_t * vtp ) [inline], [static]
```

Returns `true` if the specified timer is armed.

Precondition

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

Parameters

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

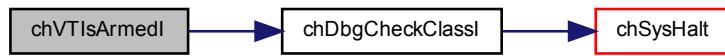
Returns

`true` if the timer is armed.

Function Class:

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:



7.22.2.18 chVTIsArmed()

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

Returns `true` if the specified timer is armed.

Precondition

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

Parameters

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

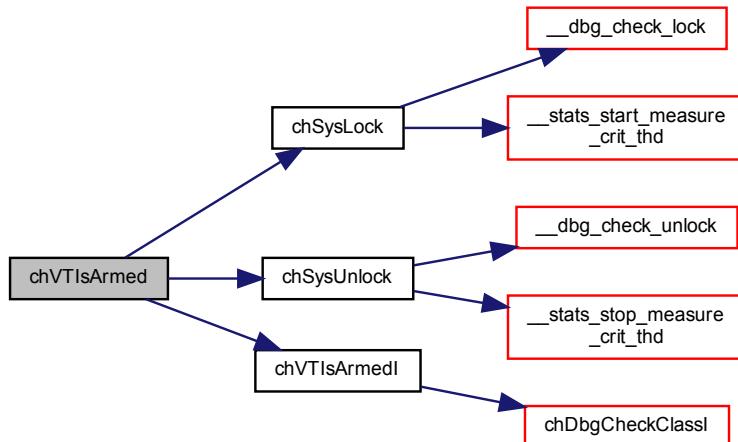
Returns

`true` if the timer is armed.

Function Class:

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

Here is the call graph for this function:

**7.22.2.19 chVTResetI()**

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

Disables a Virtual Timer.

Note

The timer is first checked and disabled only if armed.

Precondition

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

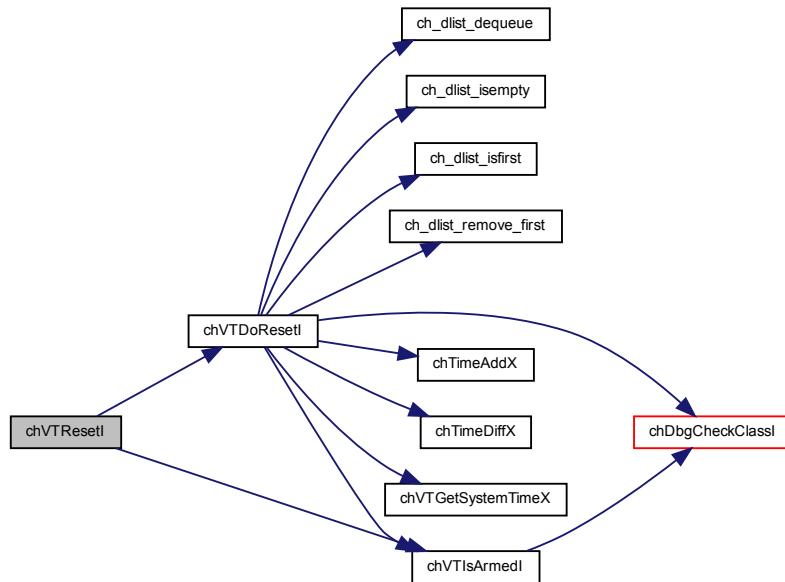
Parameters

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

Function Class:

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

Here is the call graph for this function:



7.22.2.20 `chVTReset()`

```
static void chVTReset (
    virtual_timer_t * vtp ) [inline], [static]
```

Disables a Virtual Timer.

Note

The timer is first checked and disabled only if armed.

Precondition

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

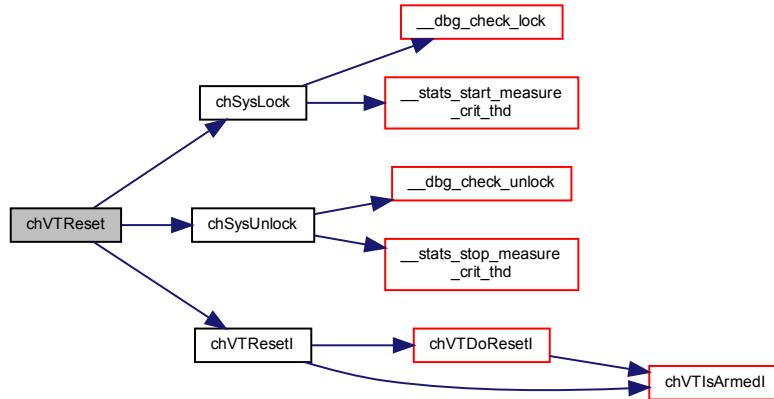
Parameters

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

Function Class:

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

Here is the call graph for this function:



7.22.2.21 chVTSetI()

```

static void chVTSetI (
    virtual_timer_t * vtp,
    sysinterval_t delay,
    vfunc_t vfunc,
    void * par ) [inline], [static]
  
```

Enables a one-shot virtual timer.

If the virtual timer was already enabled then it is re-enabled using the new parameters.

Precondition

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

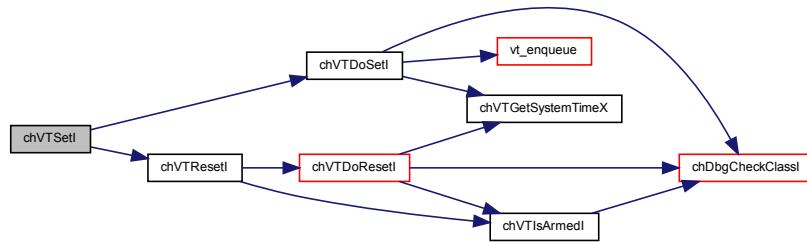
Parameters

in	<code>vtp</code>	the <code>virtual_timer_t</code> structure pointer
in	<code>delay</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> is allowed but interpreted as a normal time specification. <code>TIME_IMMEDIATE</code> this value is not allowed.
in	<code>vfunc</code>	the timer callback function. After invoking the callback the timer is disabled and the structure can be disposed or reused.
in	<code>par</code>	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:



7.22.2.22 chVTSet()

```
static void chVTSet (
    virtual_timer_t * vtp,
    sysinterval_t delay,
    vtfunc_t vtfunc,
    void * par ) [inline], [static]
```

Enables a one-shot virtual timer.

If the virtual timer was already enabled then it is re-enabled using the new parameters.

Precondition

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

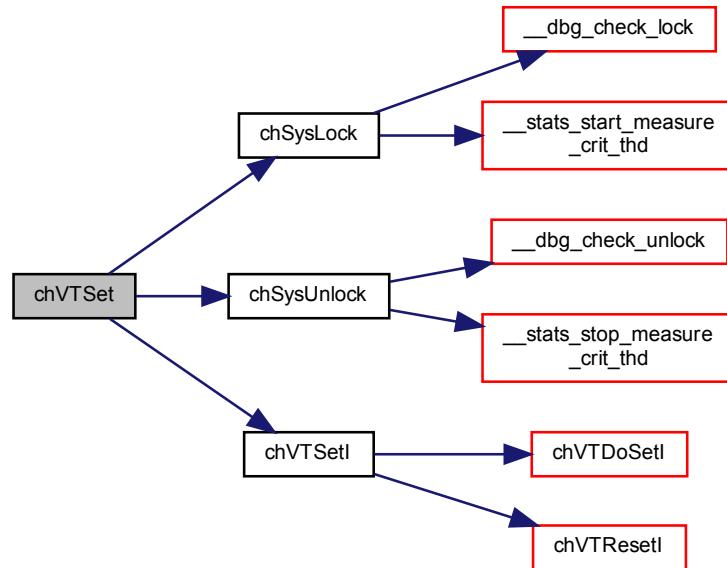
Parameters

in	<i>vtp</i>	the <code>virtual_timer_t</code> structure pointer
in	<i>delay</i>	the number of ticks before the operation timeouts, the special values are handled as follow: <ul style="list-style-type: none"> <i>TIME_INFINITE</i> is allowed but interpreted as a normal time specification. <i>TIME_IMMEDIATE</i> 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:

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:



7.22.2.23 chVTSetContinuousI()

```

static void chVTSetContinuousI (
    virtual_timer_t * vtp,
    sysinterval_t delay,
    vtfunc_t vtfunc,
    void * par ) [inline], [static]
  
```

Enables a continuous virtual timer.

If the virtual timer was already enabled then it is re-enabled using the new parameters.

Precondition

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

Parameters

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

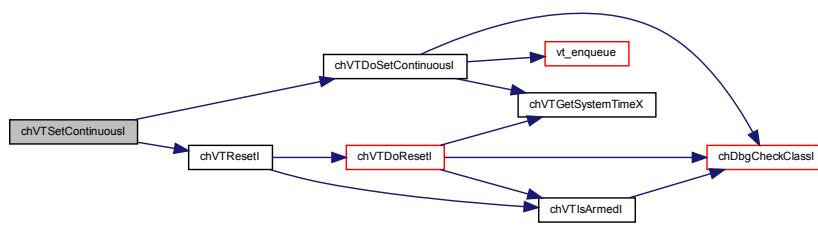
Parameters

in	<i>delay</i>	the number of ticks before the operation timeouts, the special values are handled as follow: <ul style="list-style-type: none"> • <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:

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:



7.22.2.24 chVTSetContinuous()

```

static void chVTSetContinuous (
    virtual_timer_t * vtp,
    sysinterval_t delay,
    vfunc_t vfunc,
    void * par ) [inline], [static]
  
```

Enables a continuous virtual timer.

If the virtual timer was already enabled then it is re-enabled using the new parameters.

Precondition

The timer must have been initialized using [chVTOBJECTInit\(\)](#) or [chVTDoSetI\(\)](#).

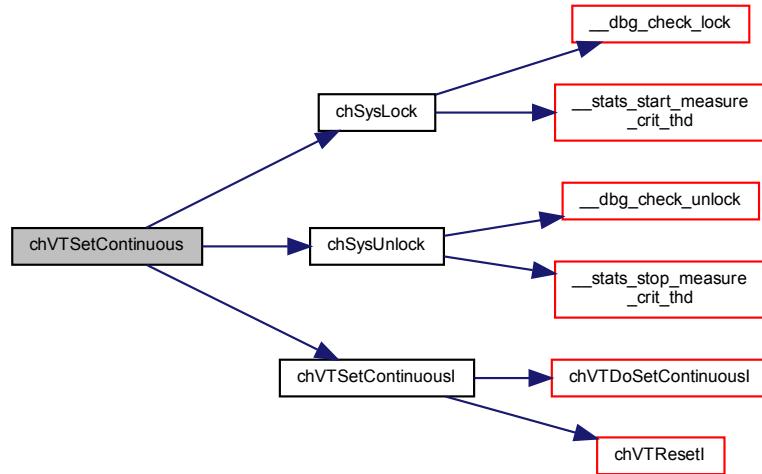
Parameters

in	<i>vtp</i>	the <code>virtual_timer_t</code> structure pointer
in	<i>delay</i>	the number of ticks before the operation timeouts, the special values are handled as follow: <ul style="list-style-type: none"> • <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.
		Chibios/RT

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:

**7.22.2.25 chVTGetReloadIntervalX()**

```
static sysinterval_t chVTGetReloadIntervalX (
    virtual_timer_t * vtp ) [inline], [static]
```

Returns the current reload value.

Parameters

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

Returns

The reload value.

Function Class:

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

7.22.2.26 chVTSetReloadIntervalX()

```
static void chVTSetReloadIntervalX (
    virtual_timer_t * vtp,
    sysinterval_t reload ) [inline], [static]
```

Changes a timer reload time interval.

Note

This function is meant to be called from a timer callback, it does nothing in any other context.

Calling this function from a one-shot timer callback turns it into a continuous timer.

Parameters

in	<i>vtp</i>	the <code>virtual_timer_t</code> structure pointer
in	<i>reload</i>	the new reload value, zero means no reload

Function Class:

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

7.22.2.27 chVTGetTimeStamp()

```
static systimestamp_t chVTGetTimeStamp (
    void ) [inline], [static]
```

Generates a monotonic time stamp.

This function generates a monotonic time stamp synchronized with the system time. The time stamp has the same resolution of system time.

Note

There is an assumption, this function must be called at least once before the system time wraps back to zero or synchronization is lost. You may use a periodic virtual timer with a very large interval in order to keep time stamps synchronized by calling this function.

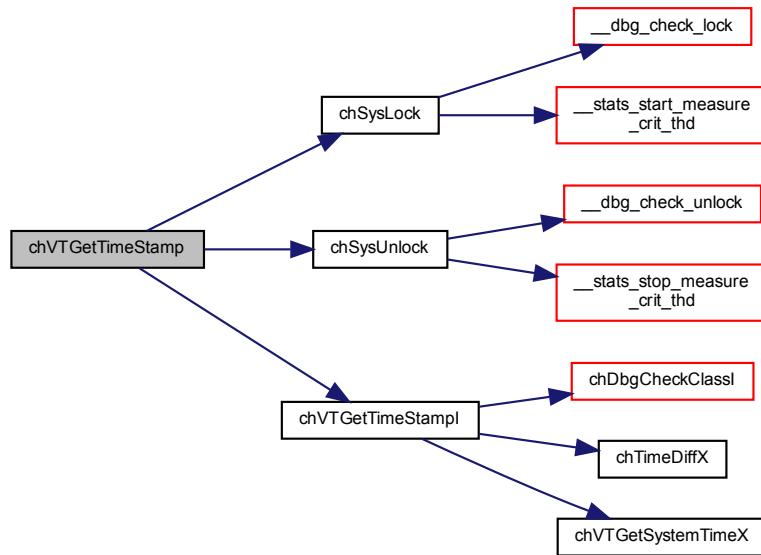
Returns

The time stamp.

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:



7.22.2.28 chVTResetTimeStamp()

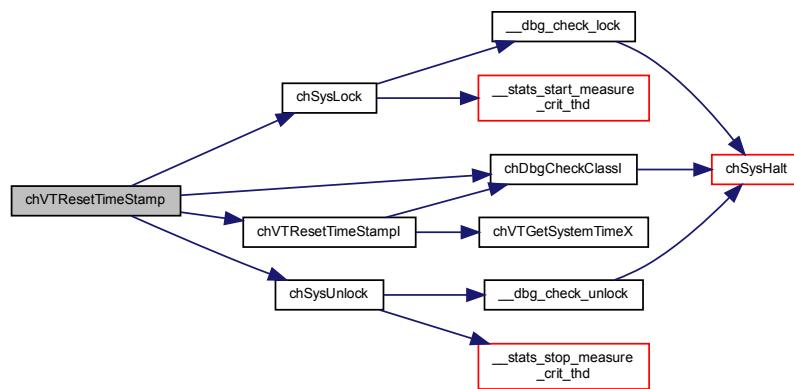
```
static void chVTResetTimeStamp (
    void ) [inline], [static]
```

Resets and re-synchronizes the time stamps monotonic counter.

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:

**7.22.2.29 __vt_object_init()**

```
static void __vt_object_init (
    virtual_timers_list_t * vtlp ) [inline], [static]
```

Virtual Timers instance initialization.

Note

Internal use only.

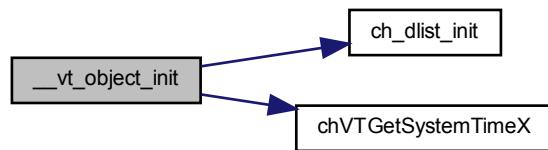
Parameters

out	<i>vtlp</i>	pointer to the <code>virtual_timers_list_t</code> structure
-----	-------------	---

Function Class:

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

Here is the call graph for this function:



7.23 Threads

7.23.1 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.

Threads queues

- `#define __THREADS_QUEUE_DATA(name) {__CH_QUEUE_DATA(name)}`
Data part of a static threads queue object initializer.
- `#define THREADS_QUEUE_DECL(name) threads_queue_t name = __THREADS_QUEUE_DATA(name)`
Static threads queue object initializer.

Working Areas

- `#define THD_WORKING_AREA_SIZE(n) MEM_ALIGN_NEXT(sizeof(thread_t) + PORT_WA_SIZE(n), P←ORT_STACK_ALIGN)`
Calculates the total Working Area size.
- `#define THD_WORKING_AREA(s, n) PORT_WORKING_AREA(s, n)`
Static working area allocation.
- `#define THD_WORKING_AREA_BASE(s) ((stkalign_t *)(s))`
Base of a working area casted to the correct type.
- `#define THD_WORKING_AREA_END(s)`
End of a working area casted to the correct type.

Threads abstraction macros

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

Threads initializers

- `#define THD_DESCRIPTOR(name, wbase, wend, prio, funcp, arg)`
Thread descriptor initializer with no affinity.
- `#define THD_DESCRIPTOR_AFFINITY(name, wbase, wend, prio, funcp, arg, oip)`
Thread descriptor initializer with no affinity.

Macro Functions

- `#define chThdSleepSeconds(sec) chThdSleep(TIME_S2I(sec))`
Delays the invoking thread for the specified number of seconds.
- `#define chThdSleepMilliseconds(msec) chThdSleep(TIME_MS2I(msec))`
Delays the invoking thread for the specified number of milliseconds.
- `#define chThdSleepMicroseconds(usec) chThdSleep(TIME_US2I(usec))`
Delays the invoking thread for the specified number of microseconds.

Typedefs

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

Data Structures

- `struct thread_descriptor_t`
Type of a thread descriptor.

Functions

- `thread_t * __thd_object_init (os_instance_t *oip, thread_t *tp, const char *name, tprio_t prio)`
Initializes a thread structure.
- `void __thd_memfill (uint8_t *startp, uint8_t *endp, uint8_t v)`
Memory fill utility.
- `thread_t * chThdCreateSuspendedI (const thread_descriptor_t *tdp)`
Creates a new thread into a static memory area.
- `thread_t * chThdCreateSuspended (const thread_descriptor_t *tdp)`
Creates a new thread into a static memory area.
- `thread_t * chThdCreateI (const thread_descriptor_t *tdp)`
Creates a new thread into a static memory area.
- `thread_t * chThdCreate (const thread_descriptor_t *tdp)`
Creates a new thread into a static memory area.
- `thread_t * chThdCreateStatic (void *wsp, size_t size, tprio_t prio, tfunc_t pf, void *arg)`
Creates a new thread into a static memory area.

- **thread_t * chThdStart (thread_t *tp)**
Resumes a thread created with `chThdCreateI()`.
- **thread_t * chThdAddRef (thread_t *tp)**
Adds a reference to a thread object.
- **void chThdRelease (thread_t *tp)**
Releases a reference to a thread object.
- **void chThdExit (msg_t msg)**
Terminates the current thread.
- **void chThdExitS (msg_t msg)**
Terminates the current thread.
- **msg_t chThdWait (thread_t *tp)**
Blocks the execution of the invoking thread until the specified thread terminates then the exit code is returned.
- **tprio_t chThdSetPriority (tprio_t newprio)**
Changes the running thread priority level then reschedules if necessary.
- **void chThdTerminate (thread_t *tp)**
Requests a thread termination.
- **void chThdSleep (sysinterval_t time)**
Suspends the invoking thread for the specified time.
- **void chThdSleepUntil (systime_t time)**
Suspends the invoking thread until the system time arrives to the specified value.
- **systime_t chThdSleepUntilWindowed (systime_t prev, systime_t next)**
Suspends the invoking thread until the system time arrives to the specified value.
- **void chThdYield (void)**
Yields the time slot.
- **msg_t chThdSuspendS (thread_reference_t *trp)**
Sends the current thread sleeping and sets a reference variable.
- **msg_t chThdSuspendTimeoutS (thread_reference_t *trp, sysinterval_t timeout)**
Sends the current thread sleeping and sets a reference variable.
- **void chThdResumeI (thread_reference_t *trp, msg_t msg)**
Wakes up a thread waiting on a thread reference object.
- **void chThdResumeS (thread_reference_t *trp, msg_t msg)**
Wakes up a thread waiting on a thread reference object.
- **void chThdResume (thread_reference_t *trp, msg_t msg)**
Wakes up a thread waiting on a thread reference object.
- **msg_t chThdEnqueueTimeoutS (threads_queue_t *tqp, sysinterval_t timeout)**
Enqueues the caller thread on a threads queue object.
- **void chThdDequeueNextI (threads_queue_t *tqp, msg_t msg)**
Dequeues and wakes up one thread from the threads queue object, if any.
- **void chThdDequeueAllI (threads_queue_t *tqp, msg_t msg)**
Dequeues and wakes up all threads from the threads queue object.
- **static thread_t * chThdGetSelfX (void)**
Returns a pointer to the current `thread_t`.
- **static tprio_t chThdGetPriorityX (void)**
Returns the current thread priority.
- **static systime_t chThdGetTicksX (thread_t *tp)**
Returns the number of ticks consumed by the specified thread.
- **static stkalign_t * chThdGetWorkingAreaX (thread_t *tp)**
Returns the working area base of the specified thread.
- **static bool chThdTerminatedX (thread_t *tp)**
Verifies if the specified thread is in the `CH_STATE_FINAL` state.
- **static bool chThdShouldTerminateX (void)**

- static `thread_t * chThdStartI (thread_t *tp)`
Resumes a thread created with `chThdCreateI()`.
- static void `chThdSleepS (sysinterval_t ticks)`
Suspends the invoking thread for the specified number of ticks.
- static void `chThdQueueObjectInit (threads_queue_t *tqp)`
Initializes a threads queue object.
- static bool `chThdQueueIsEmpty (threads_queue_t *tqp)`
Evaluates to `true` if the specified queue is empty.
- static void `chThdDoDequeueNextI (threads_queue_t *tqp, msg_t msg)`
Dequeues and wakes up one thread from the threads queue object.

7.23.2 Macro Definition Documentation

7.23.2.1 __THREADS_QUEUE_DATA

```
#define __THREADS_QUEUE_DATA(
    name ) { __CH_QUEUE_DATA (name) }
```

Data part of a static threads queue object initializer.

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

Parameters

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

7.23.2.2 THREADS_QUEUE_DECL

```
#define THREADS_QUEUE_DECL (
    name ) threads_queue_t name = __THREADS_QUEUE_DATA (name)
```

Static threads queue object initializer.

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

Parameters

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

7.23.2.3 THD_WORKING_AREA_SIZE

```
#define THD_WORKING_AREA_SIZE(
    n ) MEM_ALIGN_NEXT(sizeof(thread_t) + PORT_WA_SIZE(n), PORT_STACK_ALIGN)
```

Calculates the total Working Area size.

Parameters

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

Returns

The total used memory in bytes.

Function Class:

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

7.23.2.4 THD_WORKING_AREA

```
#define THD_WORKING_AREA(
    s,
    n ) PORT_WORKING_AREA(s, n)
```

Static working area allocation.

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

Parameters

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

Function Class:

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

7.23.2.5 THD_WORKING_AREA_BASE

```
#define THD_WORKING_AREA_BASE(
    s ) ((stkalign_t *) (s))
```

Base of a working area casted to the correct type.

Parameters

in	s	name of the working area
----	---	--------------------------

7.23.2.6 THD_WORKING_AREA_END

```
#define THD_WORKING_AREA_END (
    s )
```

Value:

```
(THD_WORKING_AREA_BASE(s) +
(sizeof (s) / sizeof (stkalign_t))) \
```

End of a working area casted to the correct type.

Parameters

in	s	name of the working area
----	---	--------------------------

7.23.2.7 THD_FUNCTION

```
#define THD_FUNCTION(
    tname,
    arg ) PORT_THD_FUNCTION(tname, arg)
```

Thread declaration macro.

Note

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

7.23.2.8 THD_DESCRIPTOR

```
#define THD_DESCRIPTOR (
    name,
    wbase,
    wend,
    prio,
    funcp,
    arg )
```

Value:

```
{ (name), \ \ }
```

```
(wbase),  
(wend),  
(prio),  
(funcp),  
(arg),  
NULL  
}
```

Thread descriptor initializer with no affinity.

Parameters

in	<i>name</i>	thread name
in	<i>wbase</i>	pointer to the working area base
in	<i>wend</i>	pointer to the working area end
in	<i>prio</i>	thread priority
in	<i>funcp</i>	thread function pointer
in	<i>arg</i>	thread argument

7.23.2.9 THD_DESCRIPTOR_AFFINITY

```
#define THD_DESCRIPTOR_AFFINITY(
    name,
    wbase,
    wend,
    prio,
    funcp,
    arg,
    oip )
```

Value:

```
{ \
    (name),
    (wbase),
    (wend),
    (prio),
    (funcp),
    (arg),
    (oip)
}
```

```
\ \
\ \
\ \
```

Thread descriptor initializer with no affinity.

Parameters

in	<i>name</i>	thread name
in	<i>wbase</i>	pointer to the working area base
in	<i>wend</i>	pointer to the working area end
in	<i>prio</i>	thread priority
in	<i>funcp</i>	thread function pointer
in	<i>arg</i>	thread argument
in	<i>oip</i>	instance affinity

7.23.2.10 chThdSleepSeconds

```
#define chThdSleepSeconds(
    sec ) chThdSleep(TIME_S2I(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.
 Use of this macro for large values is not secure because integer overflows, make sure your value can be correctly converted.

Parameters

in	<i>sec</i>	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.

7.23.2.11 chThdSleepMilliseconds

```
#define chThdSleepMilliseconds( msec ) chThdSleep(TIME_MS2I(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.
 Use of this macro for large values is not secure because integer overflows, make sure your value can be correctly converted.

Parameters

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

Function Class:

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

7.23.2.12 chThdSleepMicroseconds

```
#define chThdSleepMicroseconds( usec ) chThdSleep(TIME_US2I(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.
 Use of this macro for large values is not secure because integer overflows, make sure your value can be correctly converted.

Parameters

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

Function Class:

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

7.23.3 Typedef Documentation

7.23.3.1 tfunc_t

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

Thread function.

7.23.4 Function Documentation

7.23.4.1 __thd_object_init()

```
thread_t * __thd_object_init (
    os_instance_t * oip,
    thread_t * tp,
    const char * name,
    tprio_t prio )
```

Initializes a thread structure.

Note

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

Parameters

in	<i>oip</i>	pointer to the OS instance
in	<i>tp</i>	pointer to the thread
in	<i>name</i>	thread name
in	<i>prio</i>	the priority level for the new thread

Chibios/RT

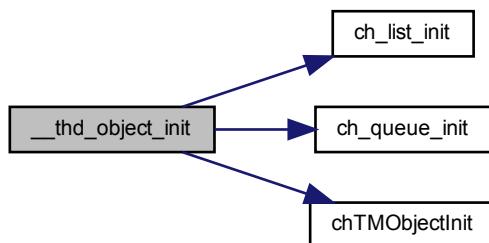
Returns

The same thread pointer passed as parameter.

Function Class:

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

Here is the call graph for this function:

**7.23.4.2 __thd_memfill()**

```
void __thd_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.

7.23.4.3 chThdCreateSuspendedI()

```
thread_t * chThdCreateSuspendedI (
    const thread_descriptor_t * tdp )
```

Creates a new thread into a static memory area.

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

Postcondition

The created thread has a reference counter set to one, it is caller responsibility to call [chThdRelease\(\)](#) or [chThdWait\(\)](#) in order to release the reference. The thread persists in the registry until its reference counter reaches zero.

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

Note

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

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

Parameters

out	tdp	pointer to the thread descriptor
-----	-----	----------------------------------

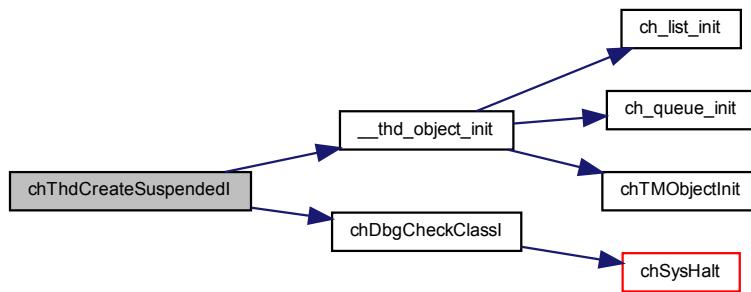
Returns

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

Function Class:

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

Here is the call graph for this function:



7.23.4.4 chThdCreateSuspended()

```
thread_t * chThdCreateSuspended (
    const thread_descriptor_t * tdp )
```

Creates a new thread into a static memory area.

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

Postcondition

The created thread has a reference counter set to one, it is caller responsibility to call `chThdRelease()` or `chThdWait()` in order to release the reference. The thread persists in the registry until its reference counter reaches zero.

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

Note

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

Parameters

out	<code>tdp</code>	pointer to the thread descriptor
-----	------------------	----------------------------------

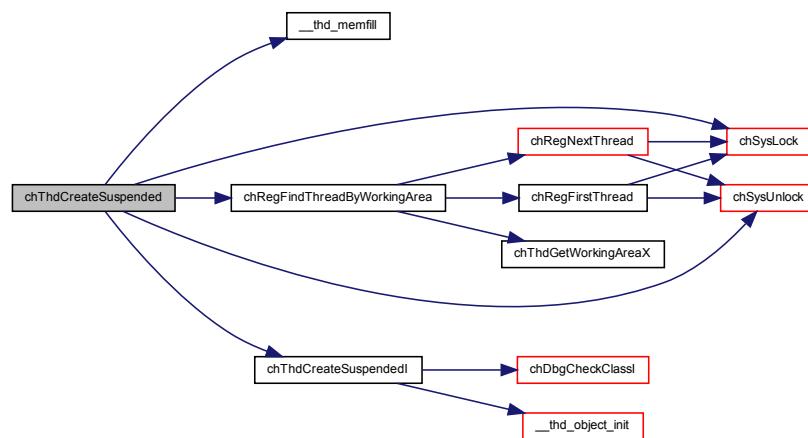
Returns

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

Function Class:

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

Here is the call graph for this function:



7.23.4.5 chThdCreateI()

```
thread_t * chThdCreateI (
    const thread_descriptor_t * tdp )
```

Creates a new thread into a static memory area.

The new thread is initialized and make ready to execute.

Postcondition

The created thread has a reference counter set to one, it is caller responsibility to call `chThdRelease()` or `chThdWait()` in order to release the reference. The thread persists in the registry until its reference counter reaches zero.

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

Note

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

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

Parameters

out	<code>tdp</code>	pointer to the thread descriptor
-----	------------------	----------------------------------

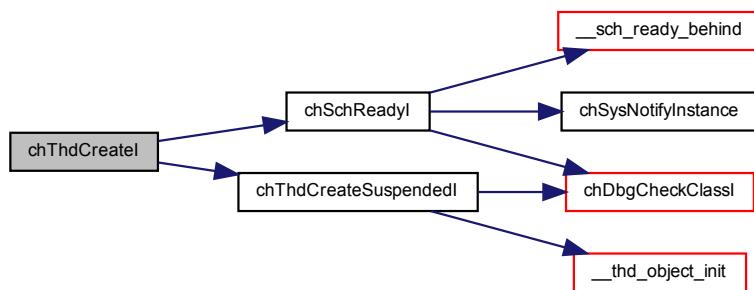
Returns

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

Function Class:

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

Here is the call graph for this function:



7.23.4.6 chThdCreate()

```
thread_t * chThdCreate (
    const thread_descriptor_t * tdp )
```

Creates a new thread into a static memory area.

The new thread is initialized and make ready to execute.

Postcondition

The created thread has a reference counter set to one, it is caller responsibility to call [chThdRelease\(\)](#) or [chThdWait\(\)](#) in order to release the reference. The thread persists in the registry until its reference counter reaches zero.

Note

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

Parameters

out	tdp	pointer to the thread descriptor
-----	-----	----------------------------------

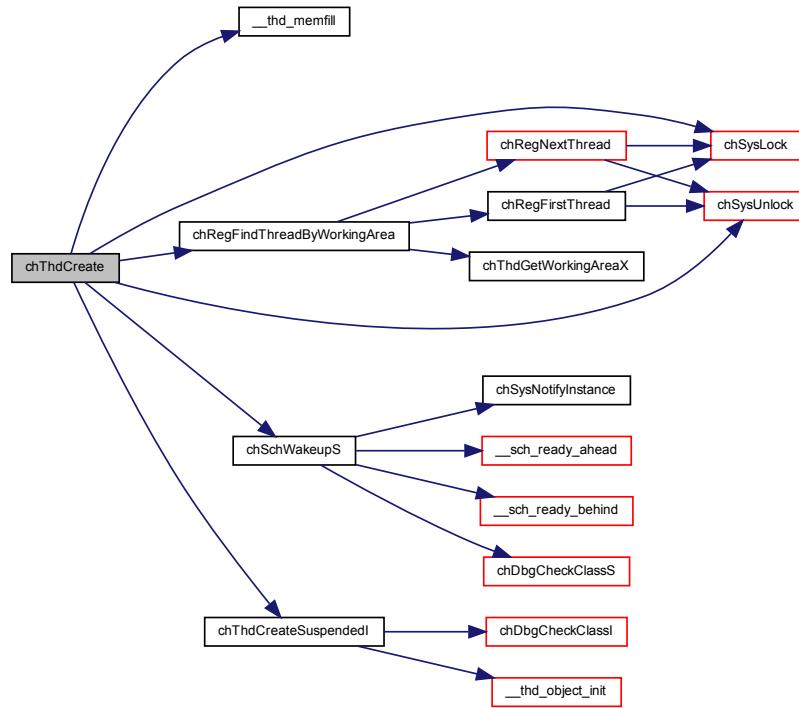
Returns

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

Function Class:

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

Here is the call graph for this function:



7.23.4.7 chThdCreateStatic()

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

Creates a new thread into a static memory area.

Postcondition

The created thread has a reference counter set to one, it is caller responsibility to call `chThdRelease()` or `chThdWait()` in order to release the reference. The thread persists in the registry until its reference counter reaches zero.

Note

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

Parameters

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

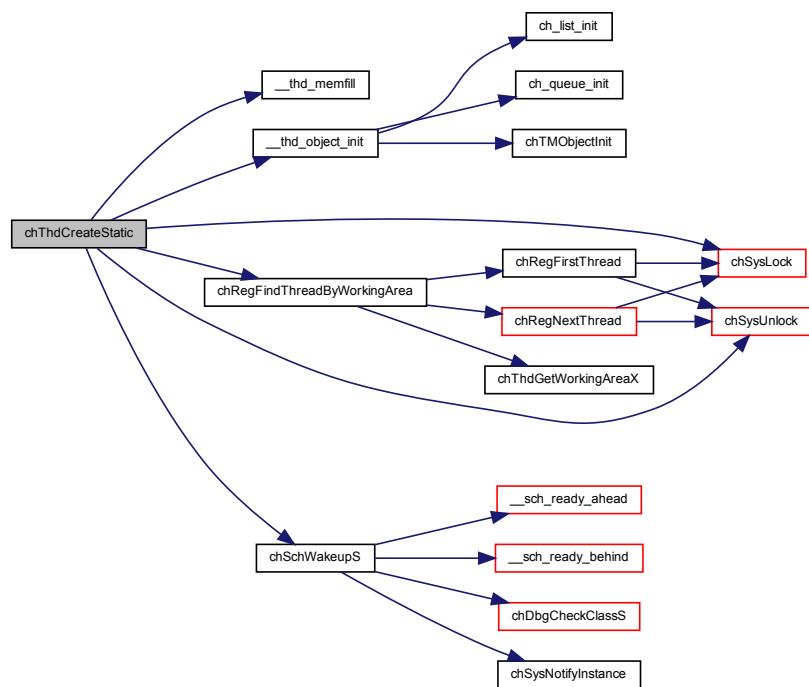
Returns

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

Function Class:

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

Here is the call graph for this function:



7.23.4.8 chThdStart()

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

Resumes a thread created with `chThdCreateI()`.

Parameters

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

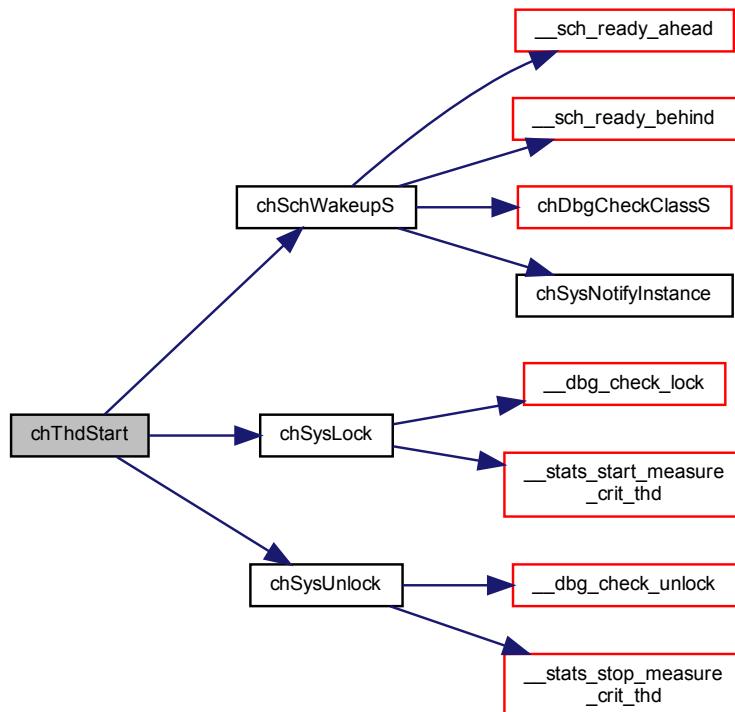
Returns

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

Function Class:

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:

7.23.4.9 `chThdAddRef()`

```
thread_t * chThdAddRef (
    thread_t * tp )
```

Adds a reference to a thread object.

Precondition

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

Parameters

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

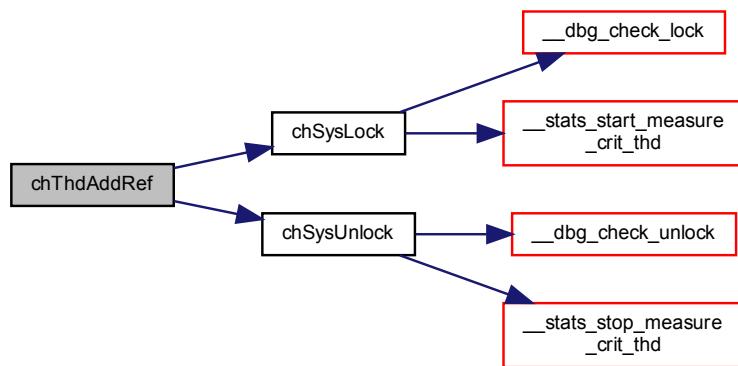
Returns

The same thread pointer passed as parameter representing the new reference.

Function Class:

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

Here is the call graph for this function:

**7.23.4.10 chThdRelease()**

```
void chThdRelease (
    thread_t * tp )
```

Releases a reference to a thread object.

If the references counter reaches zero **and** the thread is in the CH_STATE_FINAL state then the thread's memory is returned to the proper allocator and the thread is removed from the registry.

Threads whose counter reaches zero and are still active become "detached" and will be removed from registry on termination.

Precondition

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

Note

Static threads are not affected.

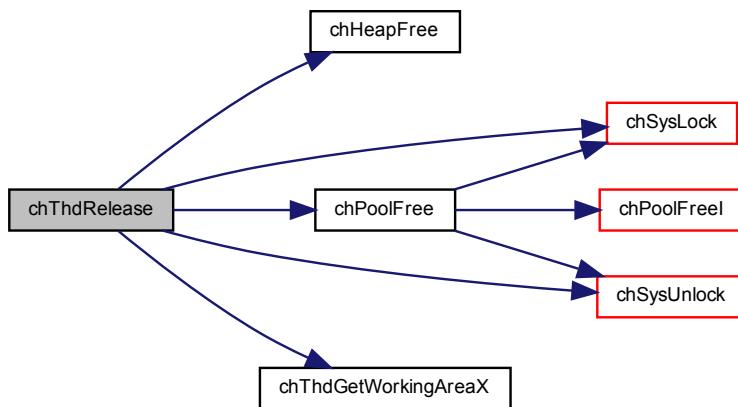
Parameters

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

Function Class:

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

Here is the call graph for this function:

7.23.4.11 `chThdExit()`

```
void chThdExit (
    msg_t msg )
```

Terminates the current thread.

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

Postcondition

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

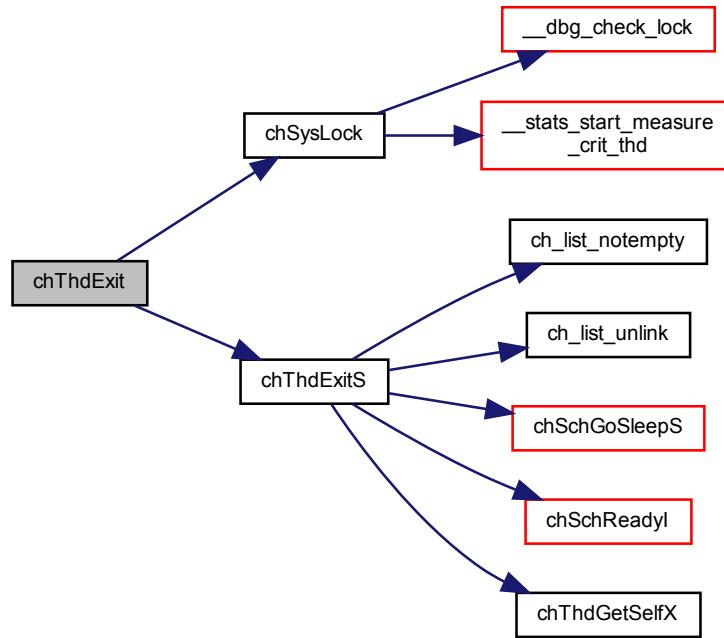
Parameters

in	<i>msg</i>	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:

**7.23.4.12 chThdExitS()**

```
void chThdExitS (
    msg_t msg )
```

Terminates the current thread.

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

Postcondition

Exiting a non-static thread that does not have references (detached) causes the thread to remain in the registry. It can only be removed by performing a registry scan operation.

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.

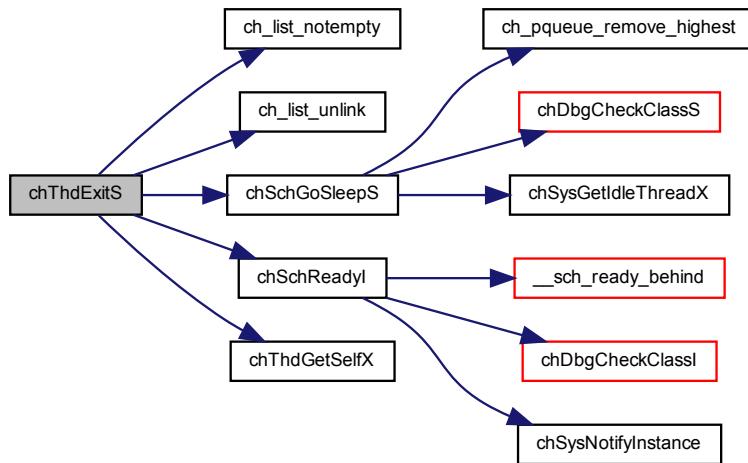
Parameters

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:



7.23.4.13 chThdWait()

```
msg_t chThdWait (
    thread_t * tp )
```

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

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

Precondition

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

Postcondition

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

Note

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

Parameters

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

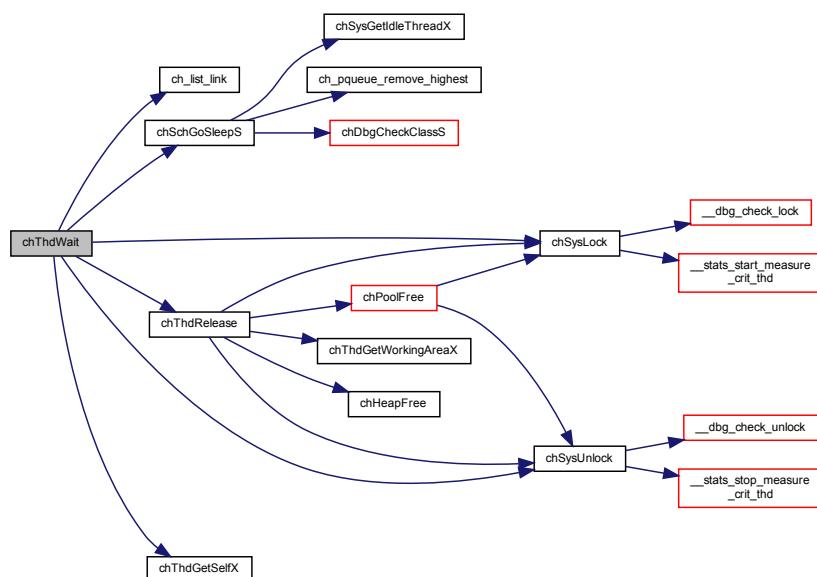
Returns

The exit code from the terminated thread.

Function Class:

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

Here is the call graph for this function:

**7.23.4.14 chThdSetPriority()**

```
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	<i>newprio</i>	the new priority level of the running thread
----	----------------	--

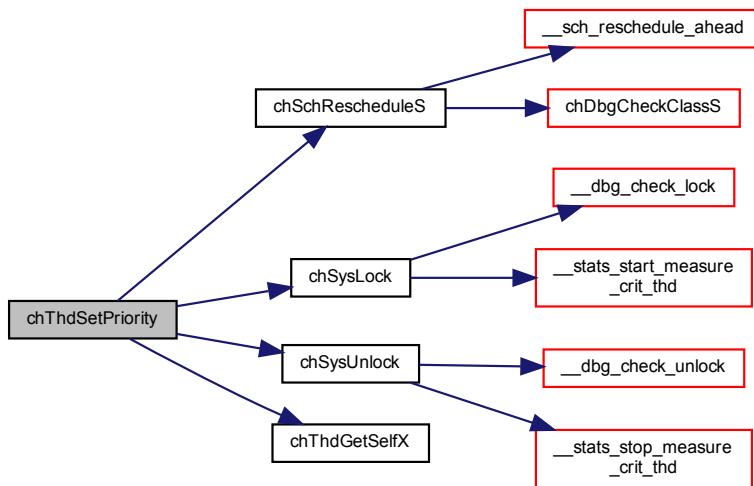
Returns

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:

**7.23.4.15 chThdTerminate()**

```
void chThdTerminate (
    thread_t * tp )
```

Requests a thread termination.

Precondition

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

Postcondition

The specified thread will terminate after detecting the termination condition.

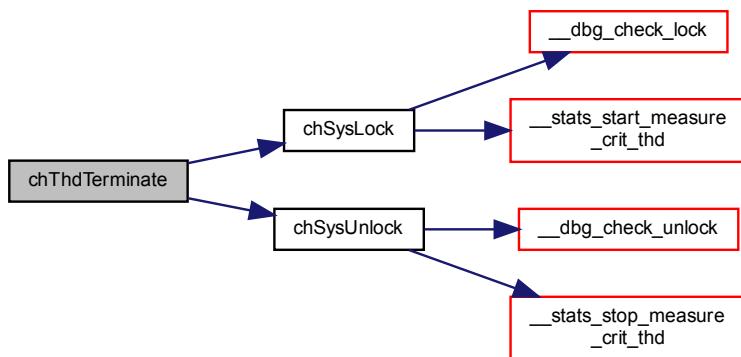
Parameters

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

Function Class:

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

Here is the call graph for this function:

**7.23.4.16 chThdSleep()**

```
void chThdSleep (
    sysinterval_t time )
```

Suspends the invoking thread for the specified time.

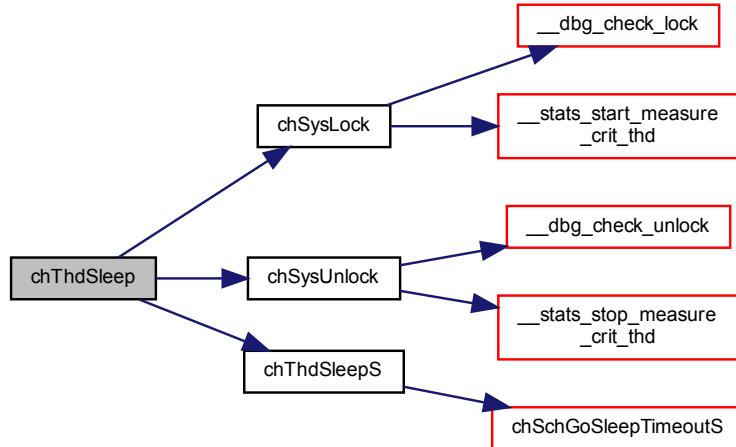
Parameters

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

Function Class:

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

Here is the call graph for this function:



7.23.4.17 chThdSleepUntil()

```
void chThdSleepUntil (
    systime_t time )
```

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

Note

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

See also

[chThdSleepUntilWindowed\(\)](#)

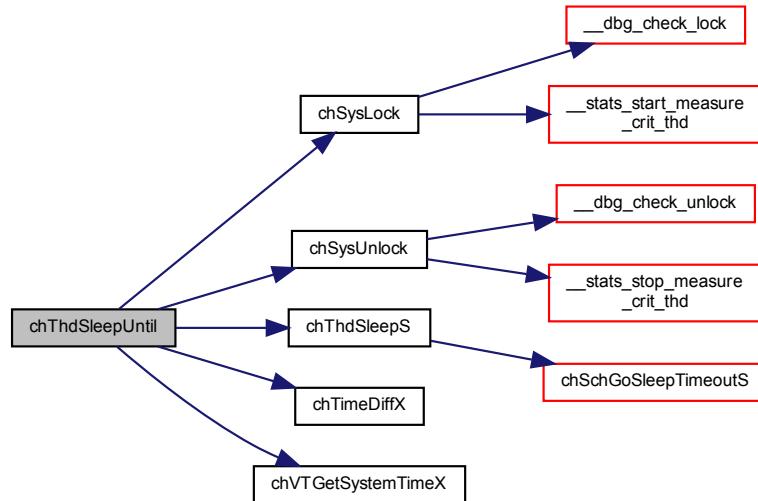
Parameters

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

Function Class:

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

Here is the call graph for this function:



7.23.4.18 chThdSleepUntilWindowed()

```

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

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

Note

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

See also

[chThdSleepUntil\(\)](#)

Parameters

in	<code>prev</code>	absolute system time of the previous deadline
in	<code>next</code>	absolute system time of the next deadline

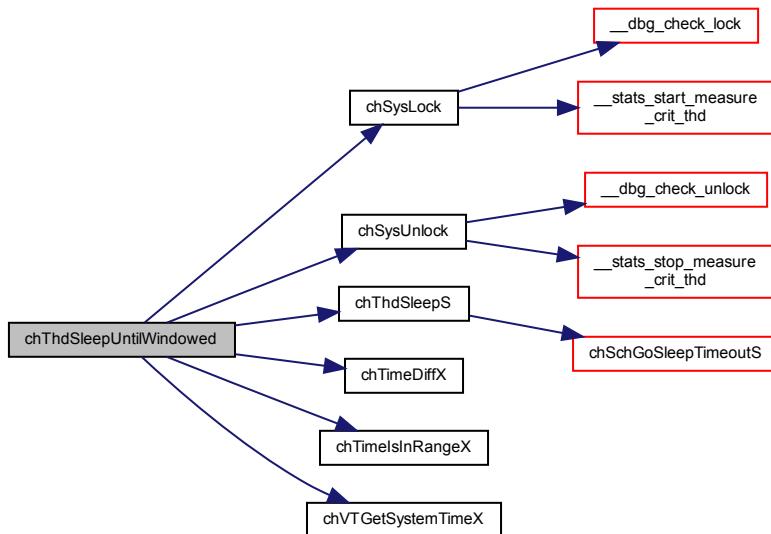
Returns

the next parameter

Function Class:

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

Here is the call graph for this function:

**7.23.4.19 chThdYield()**

```
void chThdYield (
    void )
```

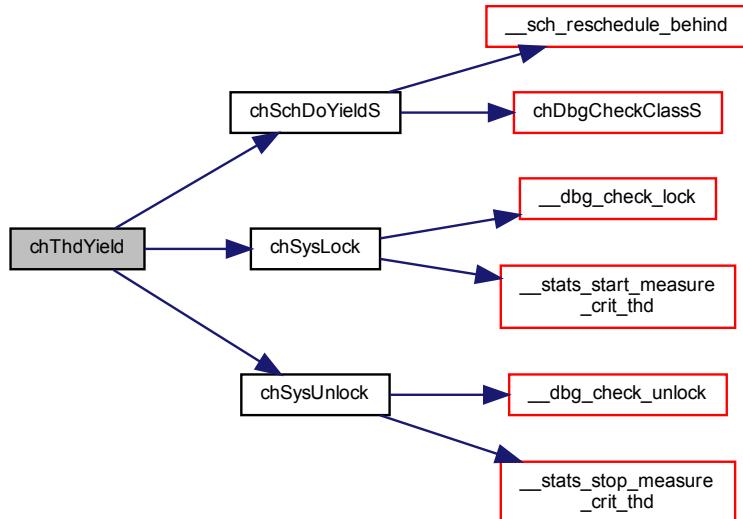
Yields the time slot.

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

Function Class:

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

Here is the call graph for this function:

**7.23.4.20 chThdSuspendS()**

```
msg_t chThdSuspendS (
    thread_reference_t * trp )
```

Sends the current thread sleeping and sets a reference variable.

Note

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

Parameters

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

Returns

The wake up message.

Function Class:

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

Here is the call graph for this function:

**7.23.4.21 chThdSuspendTimeoutS()**

```
msg_t chThdSuspendTimeoutS (
    thread_reference_t * trp,
    sysinterval_t timeout )
```

Sends the current thread sleeping and sets a reference variable.

Note

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

Parameters

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

Returns

The wake up message.

Return values

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

Function Class:

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

Here is the call graph for this function:

**7.23.4.22 chThdResumeI()**

```
void chThdResumeI (
    thread_reference_t * trp,
    msg_t msg )
```

Wakes up a thread waiting on a thread reference object.

Note

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

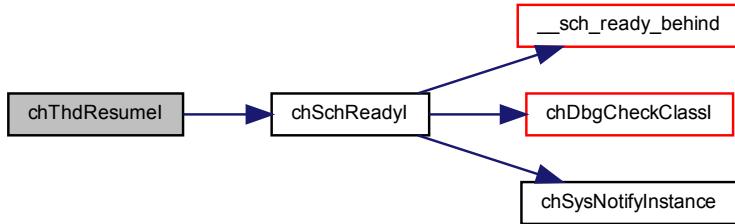
Parameters

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

Function Class:

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

Here is the call graph for this function:

**7.23.4.23 chThdResumeS()**

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

Wakes up a thread waiting on a thread reference object.

Note

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

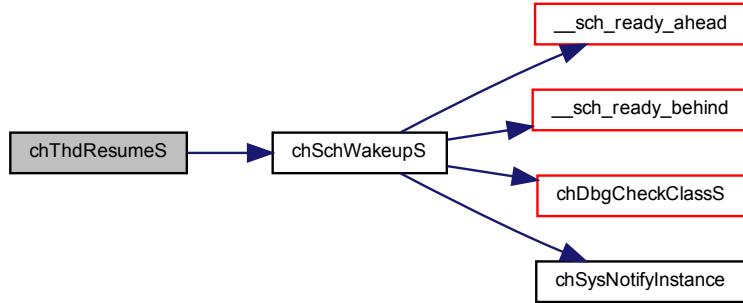
Parameters

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

Function Class:

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

Here is the call graph for this function:



7.23.4.24 chThdResume()

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

Wakes up a thread waiting on a thread reference object.

Note

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

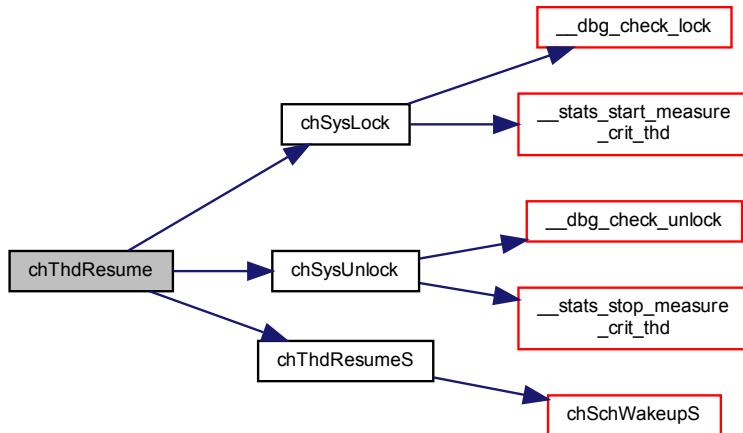
Parameters

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

Function Class:

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

Here is the call graph for this function:



7.23.4.25 chThdEnqueueTimeoutS()

```

msg_t chThdEnqueueTimeoutS (
    threads_queue_t * tqp,
    sysinterval_t timeout )
  
```

Enqueues the caller thread on a threads queue object.

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

Parameters

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

Returns

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

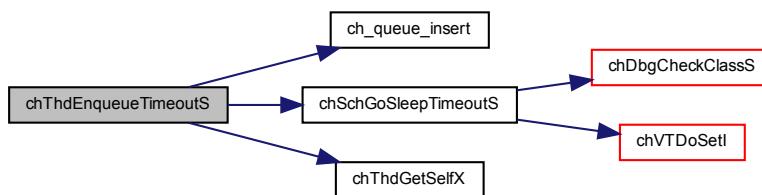
Return values

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

Function Class:

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

Here is the call graph for this function:

**7.23.4.26 chThdDequeueNextI()**

```
void chThdDequeueNextI (
    threads_queue_t * tqp,
    msg_t msg )
```

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

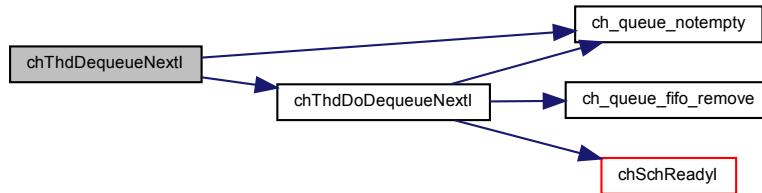
Parameters

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

Function Class:

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

Here is the call graph for this function:

**7.23.4.27 chThdDequeueAllI()**

```
void chThdDequeueAllI (
    threads_queue_t * tqp,
    msg_t msg )
```

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

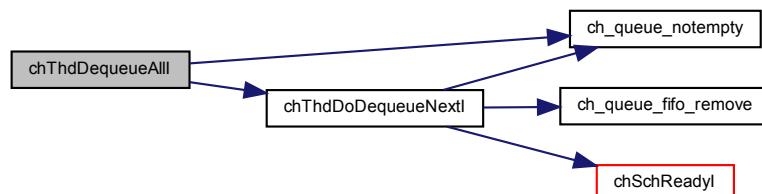
Parameters

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

Function Class:

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

Here is the call graph for this function:



7.23.4.28 chThdGetSelfX()

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

Returns a pointer to the current `thread_t`.

Returns

A pointer to the current thread.

Function Class:

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

7.23.4.29 chThdGetPriorityX()

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

Returns the current thread priority.

Note

Can be invoked in any context.

Returns

The current thread priority.

Function Class:

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

Here is the call graph for this function:



7.23.4.30 chThdGetTicksX()

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

Returns the number of ticks consumed by the specified thread.

Note

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

Parameters

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

Returns

The number of consumed system ticks.

Function Class:

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

7.23.4.31 chThdGetWorkingAreaX()

```
static stkalign_t* chThdGetWorkingAreaX (
    thread_t * tp ) [inline], [static]
```

Returns the working area base of the specified thread.

Parameters

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

Returns

The working area base pointer.

Function Class:

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

7.23.4.32 chThdTerminatedX()

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

Verifies if the specified thread is in the CH_STATE_FINAL state.

Parameters

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

Return values

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

Function Class:

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

7.23.4.33 chThdShouldTerminateX()

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

Verifies if the current thread has a termination request pending.

Return values

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

Function Class:

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

Here is the call graph for this function:

**7.23.4.34 chThdStartI()**

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

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

Parameters

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

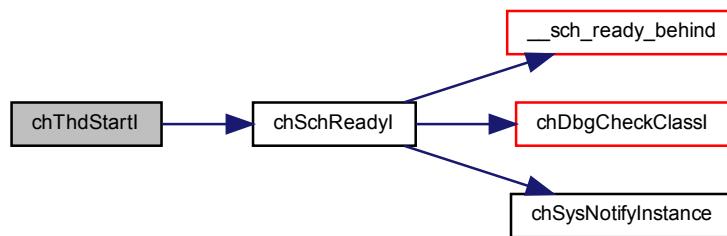
Returns

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

Function Class:

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

Here is the call graph for this function:

**7.23.4.35 chThdSleepS()**

```
static void chThdSleepS (
    sysinterval_t ticks ) [inline], [static]
```

Suspends the invoking thread for the specified number of ticks.

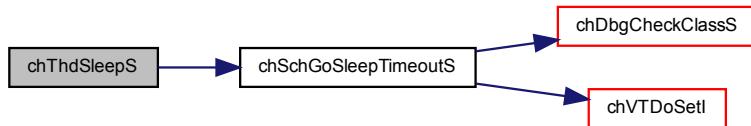
Parameters

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

Function Class:

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

Here is the call graph for this function:

**7.23.4.36 chThdQueueObjectInit()**

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

Initializes a threads queue object.

Parameters

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

Function Class:

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

Here is the call graph for this function:

**7.23.4.37 chThdQueueIsEmptyI()**

```
static bool chThdQueueIsEmptyI (
    threads_queue_t * tqp ) [inline], [static]
```

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

Parameters

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

Returns

The queue status.

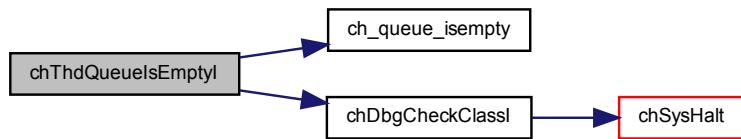
Return values

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

Function Class:

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

Here is the call graph for this function:

**7.23.4.38 chThdDoDequeueNextI()**

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

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

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

Precondition

The queue must contain at least an object.

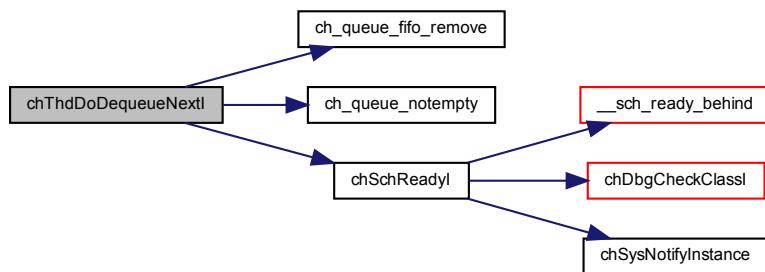
Parameters

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

Function Class:

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

Here is the call graph for this function:



7.24 Time Measurement

7.24.1 Detailed Description

Time Measurement APIs and services.

Macros

- `#define TM_CALIBRATION_LOOP 4U`
Number of iterations in the calibration loop.

Data Structures

- `struct tm_calibration_t`
Type of a time measurement calibration data.
- `struct time_measurement_t`
Type of a Time Measurement object.

Functions

- `void chTMOBJECTINIT (time_measurement_t *tmp)`
Initializes a TimeMeasurement object.
- `NOINLINE void chTMStartMeasurementX (time_measurement_t *tmp)`
Starts a measurement.
- `NOINLINE void chTMStopMeasurementX (time_measurement_t *tmp)`
Stops a measurement.
- `NOINLINE void chTMChainMeasurementToX (time_measurement_t *tmp1, time_measurement_t *tmp2)`
Stops a measurement and chains to the next one using the same time stamp.
- `static void __tm_calibration_object_init (tm_calibration_t *tcp)`
Time measurement initialization.

7.24.2 Macro Definition Documentation

7.24.2.1 TM_CALIBRATION_LOOP

```
#define TM_CALIBRATION_LOOP 4U
```

Number of iterations in the calibration loop.

Note

This is required in order to assess the best result in architectures with instruction cache.

7.24.3 Function Documentation

7.24.3.1 chTMOBJECTINIT()

```
void chTMOBJECTINIT (
    time_measurement_t * tmp )
```

Initializes a TimeMeasurement object.

Parameters

out	<i>tmp</i>	pointer to a TimeMeasurement structure
-----	------------	--

Function Class:

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

7.24.3.2 chTMStartMeasurementX()

```
NOINLINE void chTMStartMeasurementX (
    time_measurement_t * tmp )
```

Starts a measurement.

Precondition

The `time_measurement_t` structure must be initialized.

Parameters

in,out	<i>tmp</i>	pointer to a TimeMeasurement structure
--------	------------	--

Function Class:

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

7.24.3.3 chTMStopMeasurementX()

```
NOINLINE void chTMStopMeasurementX (
    time_measurement_t * tmp )
```

Stops a measurement.

Precondition

The `time_measurement_t` structure must be initialized.

Parameters

in,out	<i>tmp</i>	pointer to a <code>time_measurement_t</code> structure
--------	------------	--

Function Class:

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

7.24.3.4 chTMChainMeasurementToX()

```
NOINLINE void chTMChainMeasurementToX (
    time_measurement_t * tmp1,
    time_measurement_t * tmp2 )
```

Stops a measurement and chains to the next one using the same time stamp.

Parameters

in,out	<i>tmp1</i>	pointer to the <code>time_measurement_t</code> structure to be stopped
in,out	<i>tmp2</i>	pointer to the <code>time_measurement_t</code> structure to be started

Function Class:

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

7.24.3.5 __tm_calibration_object_init()

```
static void __tm_calibration_object_init (
    tm_calibration_t * tcp ) [inline], [static]
```

Time measurement initialization.

Note

Internal use only.

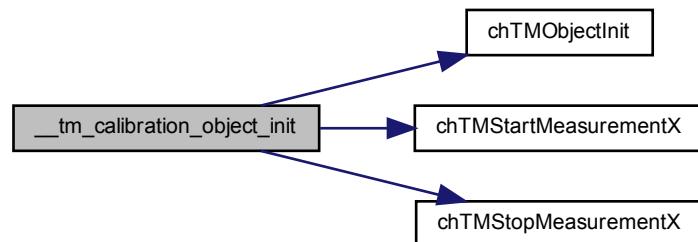
Parameters

out	<i>tcp</i>	pointer to the <code>tm_calibration_t</code> structure
-----	------------	--

Function Class:

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

Here is the call graph for this function:



7.25 Synchronization

7.25.1 Detailed Description

Synchronization services.

Modules

- Counting Semaphores
- Mutexes
- Condition Variables
- Event Flags
- Synchronous Messages

7.26 Counting Semaphores

7.26.1 Detailed Description

Semaphores related APIs and services.

Operation mode

Semaphores are a flexible synchronization primitive, ChibiOS/RT implements semaphores in their "counting semaphores" variant as defined by Edsger Dijkstra plus several enhancements like:

- Wait operation with timeout.
- Reset operation.
- Atomic wait+signal operation.
- Return message from the wait operation (OK, RESET, TIMEOUT).

The binary semaphores variant can be easily implemented using counting semaphores.

Operations defined for semaphores:

- **Signal:** The semaphore counter is increased and if the result is non-positive then a waiting thread is removed from the semaphore queue and made ready for execution.
- **Wait:** The semaphore counter is decreased and if the result becomes negative the thread is queued in the semaphore and suspended.
- **Reset:** The semaphore counter is reset to a non-negative value and all the threads in the queue are released.

Semaphores can be used as guards for mutual exclusion zones (note that mutexes are recommended for this kind of use) but also have other uses, queues guards and counters for example.

Semaphores usually use a FIFO queuing strategy but it is possible to make them order threads by priority by enabling `CH_CFG_USE_SEMAPHORES_PRIORITY` in `chconf.h`.

Precondition

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

Macros

- `#define __SEMAPHORE_DATA(name, n) {__CH_QUEUE_DATA(name.queue), n}`
Data part of a static semaphore initializer.
- `#define SEMAPHORE_DECL(name, n) semaphore_t name = __SEMAPHORE_DATA(name, n)`
Static semaphore initializer.

Typedefs

- `typedef struct ch_semaphore semaphore_t`
Semaphore structure.

Data Structures

- struct `ch_semaphore`
Semaphore structure.

Functions

- void `chSemObjectInit (semaphore_t *sp, cnt_t n)`
Initializes a semaphore with the specified counter value.
- void `chSemResetWithMessage (semaphore_t *sp, cnt_t n, msg_t msg)`
Performs a reset operation on the semaphore.
- void `chSemResetWithMessageI (semaphore_t *sp, cnt_t n, msg_t msg)`
Performs a reset operation on the semaphore.
- `msg_t chSemWait (semaphore_t *sp)`
Performs a wait operation on a semaphore.
- `msg_t chSemWaitS (semaphore_t *sp)`
Performs a wait operation on a semaphore.
- `msg_t chSemWaitTimeout (semaphore_t *sp, sysinterval_t timeout)`
Performs a wait operation on a semaphore with timeout specification.
- `msg_t chSemWaitTimeoutS (semaphore_t *sp, sysinterval_t timeout)`
Performs a wait operation on a semaphore with timeout specification.
- void `chSemSignal (semaphore_t *sp)`
Performs a signal operation on a semaphore.
- void `chSemSignall (semaphore_t *sp)`
Performs a signal operation on a semaphore.
- void `chSemAddCounterI (semaphore_t *sp, cnt_t n)`
Adds the specified value to the semaphore counter.
- `msg_t chSemSignalWait (semaphore_t *sp, semaphore_t *spw)`
Performs atomic signal and wait operations on two semaphores.
- static void `chSemReset (semaphore_t *sp, cnt_t n)`
Performs a reset operation on the semaphore.
- static void `chSemResetI (semaphore_t *sp, cnt_t n)`
Performs a reset operation on the semaphore.
- static void `chSemFastWaitI (semaphore_t *sp)`
Decreases the semaphore counter.
- static void `chSemFastSignall (semaphore_t *sp)`
Increases the semaphore counter.
- static `cnt_t chSemGetCounterI (const semaphore_t *sp)`
Returns the semaphore counter current value.

7.26.2 Macro Definition Documentation

7.26.2.1 __ SEMAPHORE _ DATA

```
#define __SEMAPHORE_DATA(
    name,
    n ) {__CH_QUEUE_DATA(name.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

7.26.2.2 SEMAPHORE_DECL

```
#define SEMAPHORE_DECL(
    name,
    n ) semaphore_t name = __SEMAPHORE_DATA(name, n)
```

Static semaphore initializer.

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

Parameters

in	<i>name</i>	the name of the semaphore variable
in	<i>n</i>	the counter initial value, this value must be non-negative

7.26.3 Typedef Documentation**7.26.3.1 semaphore_t**

```
typedef struct ch_semaphore semaphore_t
```

Semaphore structure.

7.26.4 Function Documentation**7.26.4.1 chSemObjectInit()**

```
void chSemObjectInit (
    semaphore_t * sp,
    cnt_t n )
```

Initializes a semaphore with the specified counter value.

Parameters

out	<i>sp</i>	pointer to a <code>semaphore_t</code> structure
in	<i>n</i>	initial value of the semaphore counter. Must be non-negative.

Function Class:

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

Here is the call graph for this function:



7.26.4.2 chSemResetWithMessage()

```
void chSemResetWithMessage (
    semaphore_t * sp,
    cnt_t n,
    msg_t msg )
```

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.

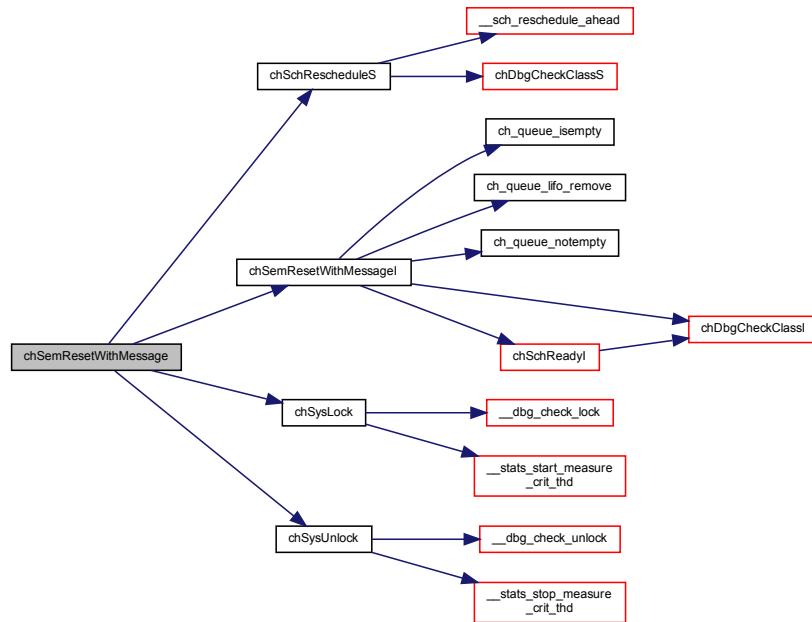
Parameters

in	<i>sp</i>	pointer to a <code>semaphore_t</code> structure
in	<i>n</i>	the new value of the semaphore counter. The value must be non-negative.
in	<i>msg</i>	message to be sent

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:



7.26.4.3 `chSemResetWithMessageI()`

```

void chSemResetWithMessageI (
    semaphore_t * sp,
    cnt_t n,
    msg_t msg )
  
```

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.

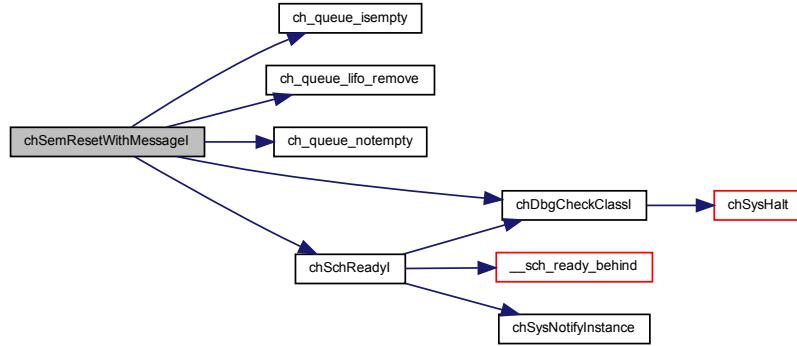
Parameters

in	<code>sp</code>	pointer to a <code>semaphore_t</code> structure
in	<code>n</code>	the new value of the semaphore counter. The value must be non-negative.
in	<code>msg</code>	message to be sent

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:

**7.26.4.4 chSemWait()**

```
msg_t chSemWait (
    semaphore_t * sp )
```

Performs a wait operation on a semaphore.

Parameters

in	<code>sp</code>	pointer to a <code>semaphore_t</code> structure
----	-----------------	---

Returns

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

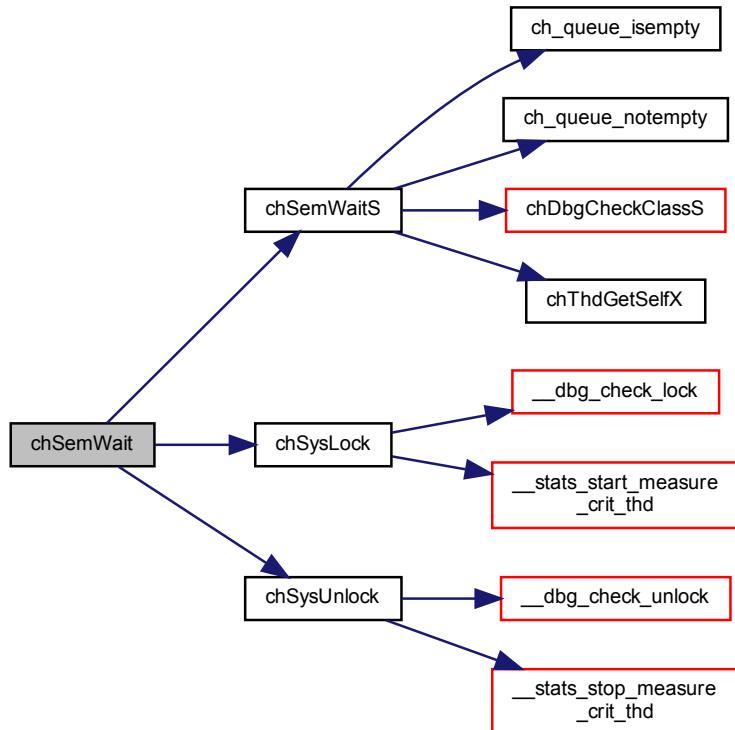
Return values

<code>MSG_OK</code>	if the thread has not stopped on the semaphore or the semaphore has been signaled.
<code>MSG_RESET</code>	if the semaphore has been reset using <code>chSemReset()</code> .

Function Class:

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

Here is the call graph for this function:



7.26.4.5 chSemWaitS()

```
msg_t chSemWaitS (
    semaphore_t * sp )
```

Performs a wait operation on a semaphore.

Parameters

in	<i>sp</i>	pointer to a <code>semaphore_t</code> structure
----	-----------	---

Returns

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

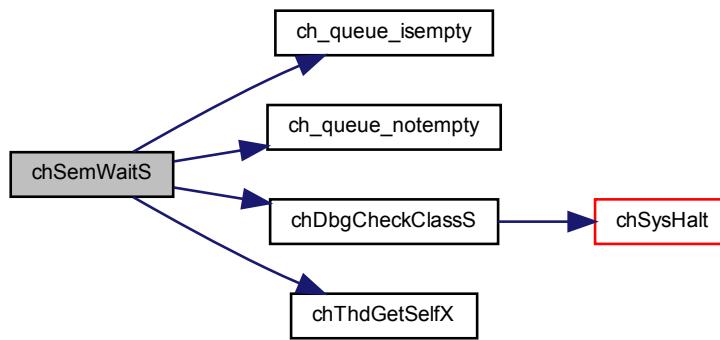
Return values

<code>MSG_OK</code>	if the thread has not stopped on the semaphore or the semaphore has been signaled.
<code>MSG_RESET</code>	if the semaphore has been reset using <code>chSemReset()</code> .

Function Class:

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:

7.26.4.6 `chSemWaitTimeout()`

```

msg_t chSemWaitTimeout (
    semaphore_t * sp,
    sysinterval_t timeout )
  
```

Performs a wait operation on a semaphore with timeout specification.

Parameters

in	<code>sp</code>	pointer to a <code>semaphore_t</code> structure
in	<code>timeout</code>	the number of ticks before the operation timeouts, the following special values are allowed: <ul style="list-style-type: none"> • <code>TIME_IMMEDIATE</code> immediate timeout. • <code>TIME_INFINITE</code> no timeout.

Returns

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

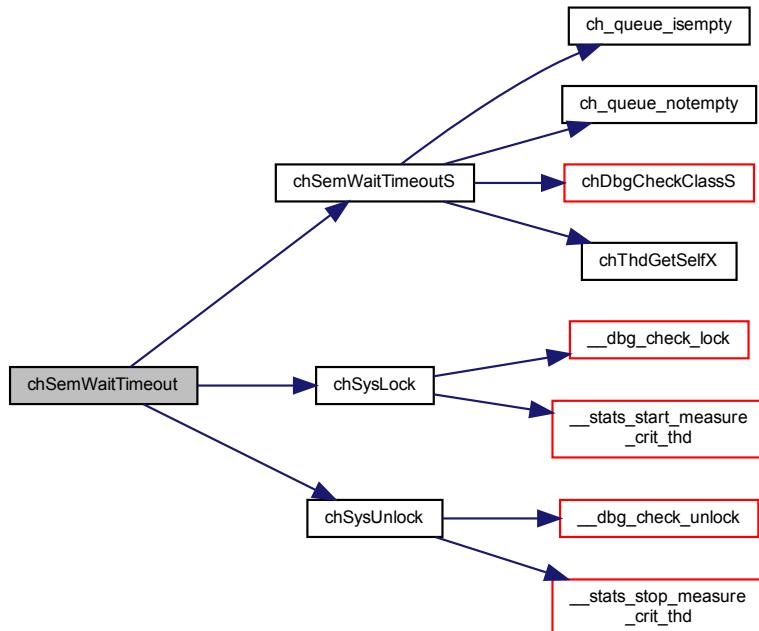
Return values

<i>MSG_OK</i>	if the thread has not stopped on the semaphore or the semaphore has been signaled.
<i>MSG_RESET</i>	if the semaphore has been reset using chSemReset () .
<i>MSG_TIMEOUT</i>	if the semaphore has not been signaled or reset within the specified timeout.

Function Class:

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

Here is the call graph for this function:

**7.26.4.7 chSemWaitTimeoutS()**

```
msg_t chSemWaitTimeoutS (
    semaphore_t * sp,
    sysinterval_t timeout )
```

Performs a wait operation on a semaphore with timeout specification.

Parameters

in	<i>sp</i>	pointer to a <code>semaphore_t</code> structure
in	<i>timeout</i>	the number of ticks before the operation timeouts, the following special values are allowed: <ul style="list-style-type: none"> • <code>TIME_IMMEDIATE</code> immediate timeout. • <code>TIME_INFINITE</code> no timeout.

Returns

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

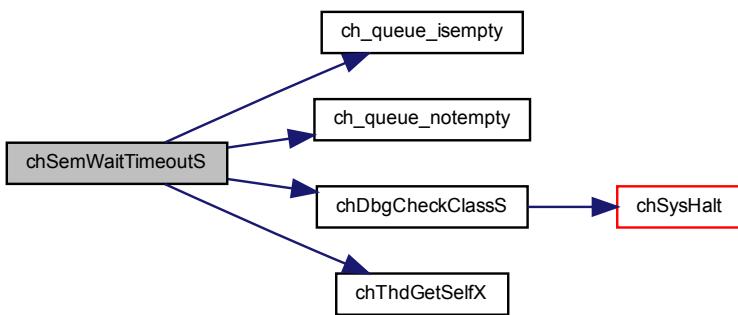
Return values

<code>MSG_OK</code>	if the thread has not stopped on the semaphore or the semaphore has been signaled.
<code>MSG_RESET</code>	if the semaphore has been reset using <code>chSemReset()</code> .
<code>MSG_TIMEOUT</code>	if the semaphore has not been signaled or reset within the specified timeout.

Function Class:

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

Here is the call graph for this function:



7.26.4.8 chSemSignal()

```
void chSemSignal (
    semaphore_t * sp )
```

Performs a signal operation on a semaphore.

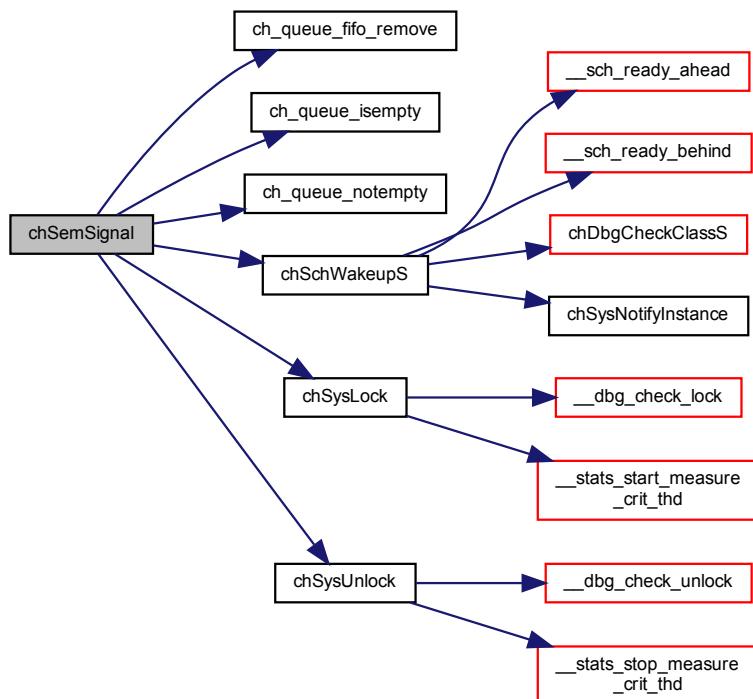
Parameters

in	<i>sp</i>	pointer to a <code>semaphore_t</code> structure
----	-----------	---

Function Class:

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

Here is the call graph for this function:

**7.26.4.9 chSemSignal()**

```
void chSemSignalI (
    semaphore_t * sp )
```

Performs a signal operation on a semaphore.

Postcondition

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

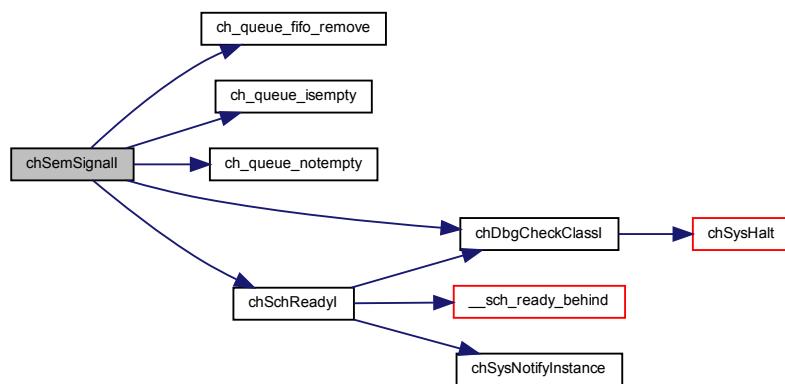
Parameters

in	<i>sp</i>	pointer to a <code>semaphore_t</code> structure
----	-----------	---

Function Class:

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

Here is the call graph for this function:



7.26.4.10 chSemAddCounterI()

```
void chSemAddCounterI (
    semaphore_t * sp,
    cnt_t n )
```

Adds the specified value to the semaphore counter.

Postcondition

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

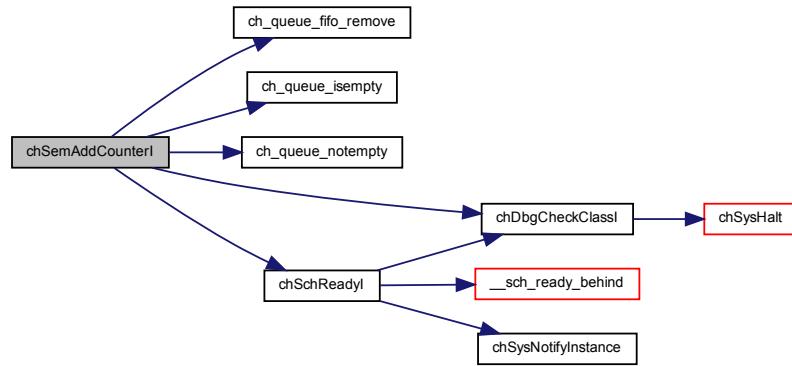
Parameters

in	<i>sp</i>	pointer to a <code>semaphore_t</code> 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:

**7.26.4.11 chSemSignalWait()**

```
msg_t chSemSignalWait (
    semaphore_t * sps,
    semaphore_t * spw )
```

Performs atomic signal and wait operations on two semaphores.

Parameters

in	<code>sps</code>	pointer to a <code>semaphore_t</code> structure to be signaled
in	<code>spw</code>	pointer to a <code>semaphore_t</code> structure to wait on

Returns

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

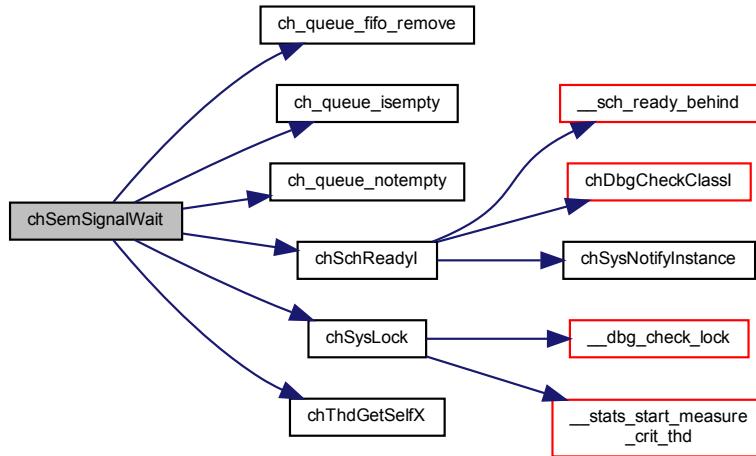
Return values

<code>MSG_OK</code>	if the thread has not stopped on the semaphore or the semaphore has been signaled.
<code>MSG_RESET</code>	if the semaphore has been reset using <code>chSemReset()</code> .

Function Class:

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

Here is the call graph for this function:

**7.26.4.12 chSemReset()**

```
static void chSemReset (
    semaphore_t * sp,
    cnt_t n ) [inline], [static]
```

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

This function implicitly sends MSG_RESET as message.

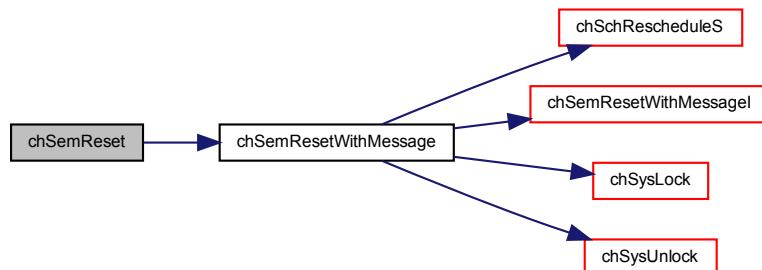
Parameters

in	<i>sp</i>	pointer to a <code>semaphore_t</code> structure
in	<i>n</i>	the new value of the semaphore counter. The value must be non-negative.

Function Class:

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

Here is the call graph for this function:

**7.26.4.13 chSemResetI()**

```
static void chSemResetI (
    semaphore_t * sp,
    cnt_t n ) [inline], [static]
```

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

This function implicitly sends MSG_RESET as message.

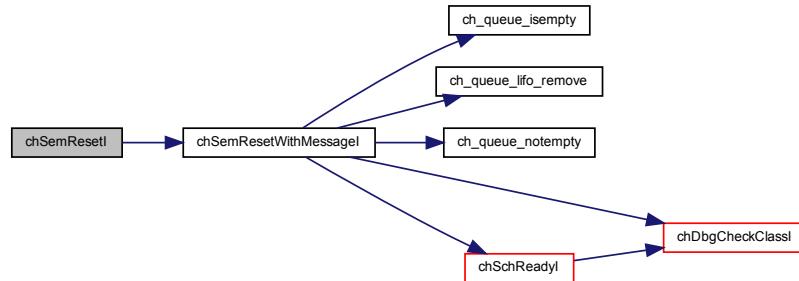
Parameters

in	<i>sp</i>	pointer to a <code>semaphore_t</code> structure
in	<i>n</i>	the new value of the semaphore counter. The value must be non-negative.

Function Class:

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

Here is the call graph for this function:

**7.26.4.14 chSemFastWaitI()**

```
static void chSemFastWaitI (
    semaphore_t * sp ) [inline], [static]
```

Decreases the semaphore counter.

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

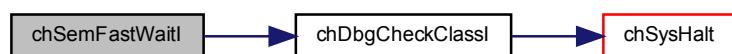
Parameters

in	<i>sp</i>	pointer to a <code>semaphore_t</code> structure
----	-----------	---

Function Class:

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

Here is the call graph for this function:



7.26.4.15 chSemFastSignal()

```
static void chSemFastSignalI (
    semaphore_t * sp ) [inline], [static]
```

Increases the semaphore counter.

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

Parameters

in	sp	pointer to a semaphore_t structure
----	----	------------------------------------

Function Class:

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

Here is the call graph for this function:



7.26.4.16 chSemGetCounterI()

```
static cnt_t chSemGetCounterI (
    const semaphore_t * sp ) [inline], [static]
```

Returns the semaphore counter current value.

Parameters

in	sp	pointer to a semaphore_t structure
----	----	------------------------------------

Returns

The semaphore counter value.

Function Class:

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

Here is the call graph for this function:



7.27 Mutexes

7.27.1 Detailed Description

Mutexes related APIs and services.

Operation mode

A mutex is a threads synchronization object that can be in two distinct states:

- Not owned (unlocked).
- Owned by a thread (locked).

Operations defined for mutexes:

- **Lock:** The mutex is checked, if the mutex is not owned by some other thread then it is associated to the locking thread else the thread is queued on the mutex in a list ordered by priority.
- **Unlock:** The mutex is released by the owner and the highest priority thread waiting in the queue, if any, is resumed and made owner of the mutex.

Constraints

In ChibiOS/RT the Unlock operations must always be performed in lock-reverse order. This restriction both improves the performance and is required for an efficient implementation of the priority inheritance mechanism. Operating under this restriction also ensures that deadlocks are no possible.

Recursive mode

By default mutexes are not recursive, this mean that it is not possible to take a mutex already owned by the same thread. It is possible to enable the recursive behavior by enabling the option `CH_CFG_USE_MUTEXES_RECURSIVE`.

The priority inversion problem

The mutexes in ChibiOS/RT implements the **full** priority inheritance mechanism in order handle the priority inversion problem.

When a thread is queued on a mutex, any thread, directly or indirectly, holding the mutex gains the same priority of the waiting thread (if their priority was not already equal or higher). The mechanism works with any number of nested mutexes and any number of involved threads. The algorithm complexity (worst case) is N with N equal to the number of nested mutexes.

Precondition

In order to use the mutex APIs the `CH_CFG_USE_MUTEXES` option must be enabled in `chconf.h`.

Postcondition

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

Macros

- `#define __MUTEX_DATA(name) {__CH_QUEUE_DATA(name.queue), NULL, NULL, 0}`
Data part of a static mutex initializer.
- `#define MUTEX_DECL(name) mutex_t name = __MUTEX_DATA(name)`
Static mutex initializer.

Typedefs

- `typedef struct ch_mutex mutex_t`
Type of a mutex structure.

Data Structures

- `struct ch_mutex`
Mutex structure.

Functions

- `void chMtxObjectInit (mutex_t *mp)`
Initializes a mutex_t structure.
- `void chMtxLock (mutex_t *mp)`
Locks the specified mutex.
- `void chMtxLockS (mutex_t *mp)`
Locks the specified mutex.
- `bool chMtxTryLock (mutex_t *mp)`
Tries to lock a mutex.
- `bool chMtxTryLockS (mutex_t *mp)`
Tries to lock a mutex.
- `void chMtxUnlock (mutex_t *mp)`
Unlocks the specified mutex.
- `void chMtxUnlockS (mutex_t *mp)`
Unlocks the specified mutex.
- `void chMtxUnlockAll (void)`
Unlocks all mutexes owned by the invoking thread.
- `void chMtxUnlockAllS (void)`
Unlocks all mutexes owned by the invoking thread.
- `static bool chMtxQueueNotEmptyS (mutex_t *mp)`
Returns true if the mutex queue contains at least a waiting thread.
- `static thread_t * chMtxGetOwnerI (mutex_t *mp)`
Returns the mutex owner thread.
- `static mutex_t * chMtxGetNextMutexX (void)`
Returns the next mutex in the mutexes stack of the current thread.

7.27.2 Macro Definition Documentation

7.27.2.1 __MUTEX_DATA

```
#define __MUTEX_DATA(  
    name ) {__CH_QUEUE_DATA(name.queue), NULL, NULL, 0}
```

Data part of a static mutex initializer.

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

Parameters

in	<i>name</i>	the name of the mutex variable
----	-------------	--------------------------------

7.27.2.2 MUTEX_DECL

```
#define MUTEX_DECL(  
    name ) mutex_t name = __MUTEX_DATA(name)
```

Static mutex initializer.

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

Parameters

in	<i>name</i>	the name of the mutex variable
----	-------------	--------------------------------

7.27.3 Typedef Documentation**7.27.3.1 mutex_t**

```
typedef struct ch_mutex mutex_t
```

Type of a mutex structure.

7.27.4 Function Documentation**7.27.4.1 chMtxObjectInit()**

```
void chMtxObjectInit (  
    mutex_t * mp )
```

Initializes a `mutex_t` structure.

Parameters

out	<i>mp</i>	pointer to a <code>mutex_t</code> structure
-----	-----------	---

Function Class:

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

Here is the call graph for this function:



7.27.4.2 chMtxLock()

```
void chMtxLock (
    mutex_t * mp )
```

Locks the specified mutex.

Postcondition

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

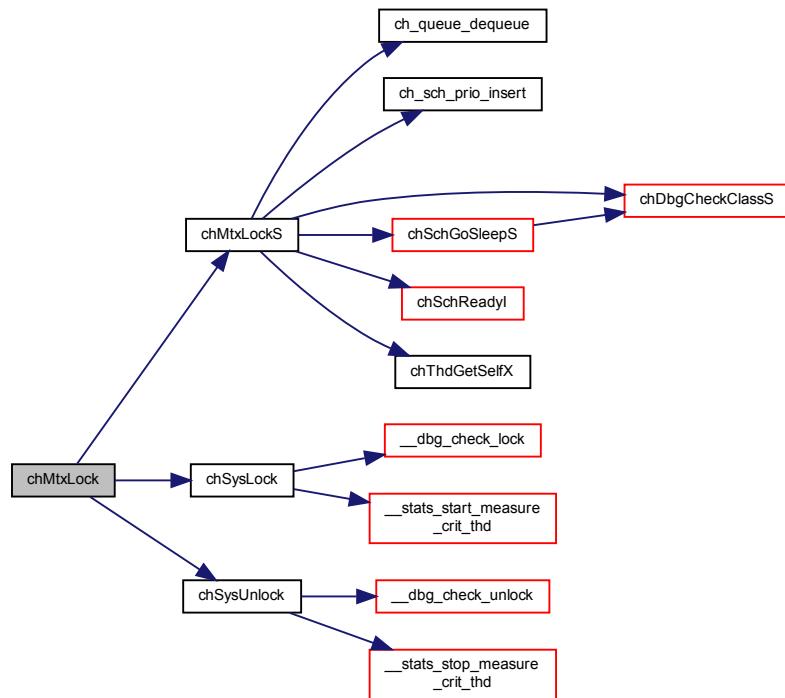
Parameters

in	mp	pointer to the mutex_t structure
----	----	----------------------------------

Function Class:

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

Here is the call graph for this function:



7.27.4.3 chMtxLockS()

```
void chMtxLockS (
    mutex_t * mp )
```

Locks the specified mutex.

Postcondition

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

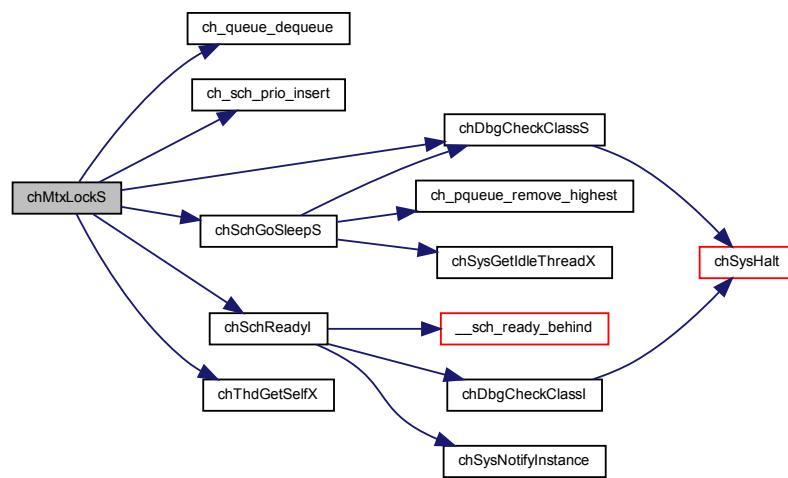
Parameters

in	<code>mp</code>	pointer to the <code>mutex_t</code> structure
----	-----------------	---

Function Class:

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

Here is the call graph for this function:



7.27.4.4 `chMtxTryLock()`

```
bool chMtxTryLock (
    mutex_t * mp )
```

Tries to lock a mutex.

This function attempts to lock a mutex, if the mutex is already locked by another thread then the function exits without waiting.

Postcondition

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

Note

This function does not have any overhead related to the priority inheritance mechanism because it does not try to enter a sleep state.

Parameters

in	<code>mp</code>	pointer to the <code>mutex_t</code> structure
----	-----------------	---

Returns

The operation status.

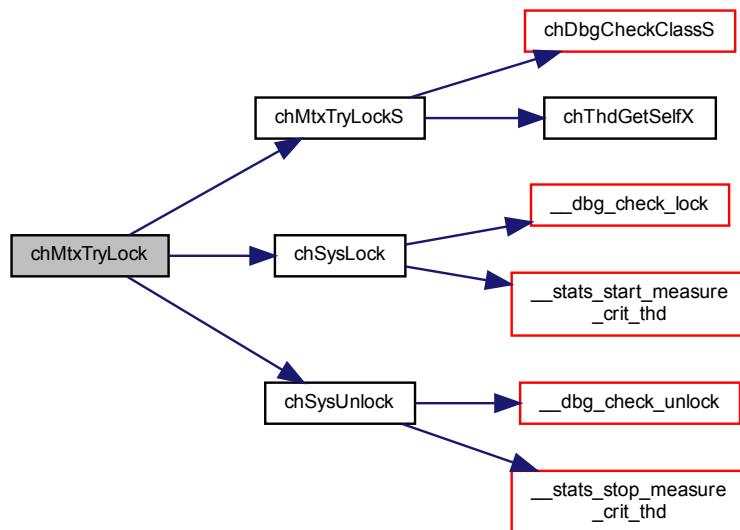
Return values

<i>true</i>	if the mutex has been successfully acquired
<i>false</i>	if the lock attempt failed.

Function Class:

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:

**7.27.4.5 chMtxTryLockS()**

```
bool chMtxTryLockS (
    mutex_t * mp )
```

Tries to lock a mutex.

This function attempts to lock a mutex, if the mutex is already taken by another thread then the function exits without waiting.

Postcondition

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

Note

This function does not have any overhead related to the priority inheritance mechanism because it does not try to enter a sleep state.

Parameters

in	<i>mp</i>	pointer to the <code>mutex_t</code> structure
----	-----------	---

Returns

The operation status.

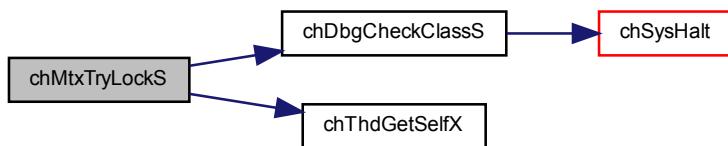
Return values

<i>true</i>	if the mutex has been successfully acquired
<i>false</i>	if the lock attempt failed.

Function Class:

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:

**7.27.4.6 chMtxUnlock()**

```
void chMtxUnlock (
    mutex_t * mp )
```

Unlocks the specified mutex.

Note

Mutexes must be unlocked in reverse lock order. Violating this rules will result in a panic if assertions are enabled.

Precondition

The invoking thread **must** have at least one owned mutex.

Postcondition

The mutex is unlocked and removed from the per-thread stack of owned mutexes.

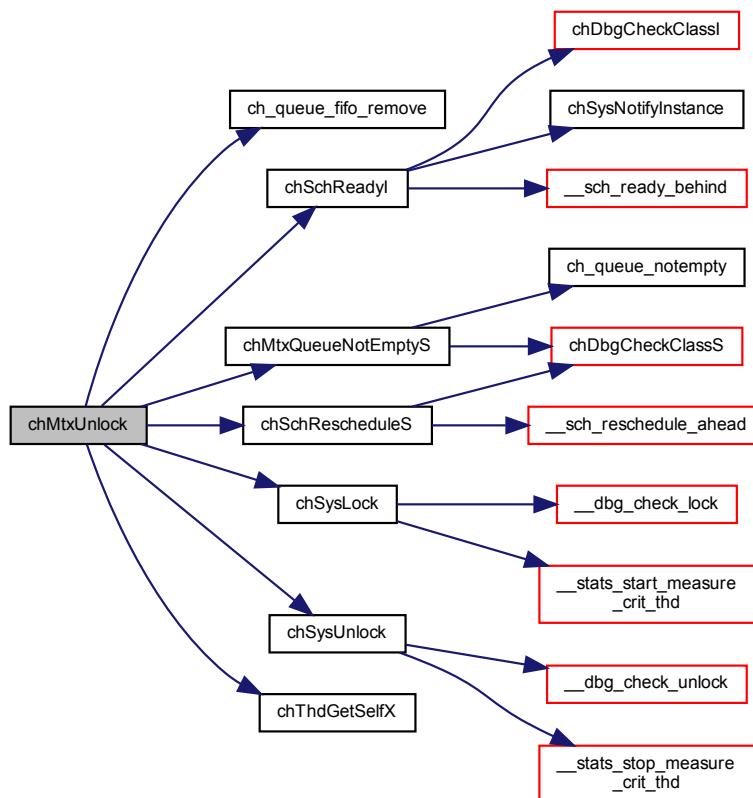
Parameters

in	<i>mp</i>	pointer to the <code>mutex_t</code> structure
----	-----------	---

Function Class:

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

Here is the call graph for this function:

7.27.4.7 `chMtxUnlockS()`

```
void chMtxUnlocks (
    mutex_t * mp )
```

Unlocks the specified mutex.

Note

Mutexes must be unlocked in reverse lock order. Violating this rules will result in a panic if assertions are enabled.

Precondition

The invoking thread **must** have at least one owned mutex.

Postcondition

The mutex is unlocked and removed from the per-thread stack of owned mutexes.

This function does not reschedule so a call to a rescheduling function must be performed before unlocking the kernel.

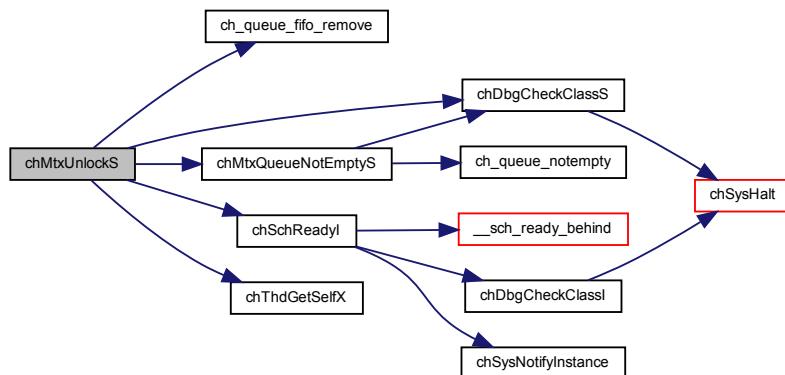
Parameters

in	<i>mp</i>	pointer to the <code>mutex_t</code> structure
----	-----------	---

Function Class:

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

Here is the call graph for this function:

**7.27.4.8 chMtxUnlockAllS()**

```
void chMtxUnlockAllS (
    void )
```

Unlocks all mutexes owned by the invoking thread.

Postcondition

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

This function does not reschedule so a call to a rescheduling function must be performed before unlocking the kernel.

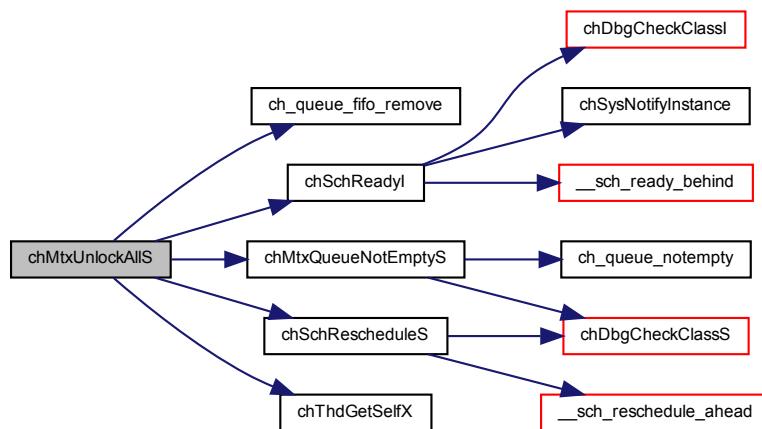
Note

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:

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:



7.27.4.9 chMtxUnlockAll()

```
void chMtxUnlockAll (
    void )
```

Unlocks all mutexes owned by the invoking thread.

Postcondition

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

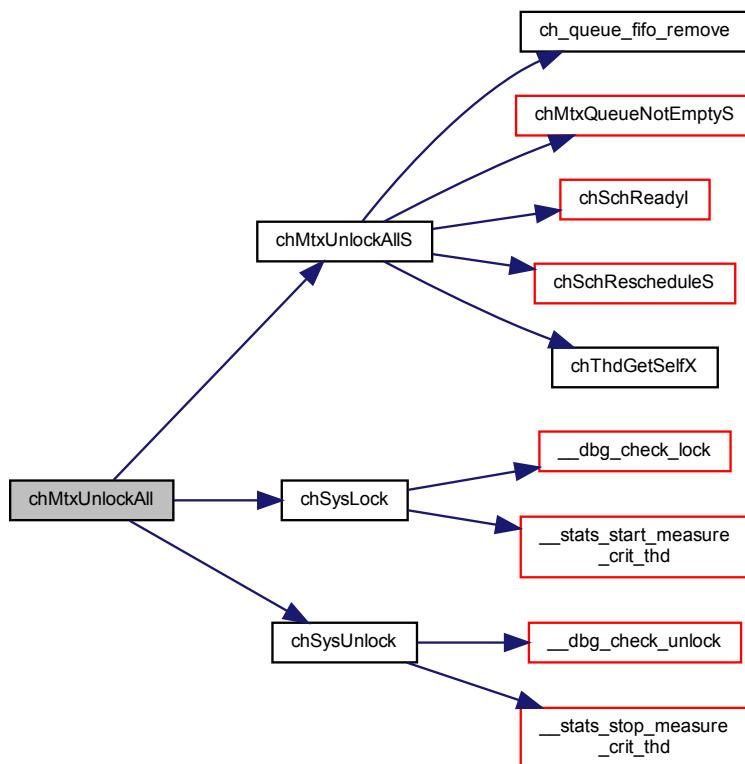
Note

This function is **MUCH MORE** efficient than releasing the mutexes one by one and not just because the call overhead, this function does not have any overhead related to the priority inheritance mechanism.

Function Class:

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

Here is the call graph for this function:

**7.27.4.10 chMtxQueueNotEmptyS()**

```
static bool chMtxQueueNotEmptyS (
    mutex_t * mp ) [inline], [static]
```

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

Parameters

out	<code>mp</code>	pointer to a <code>mutex_t</code> structure
-----	-----------------	---

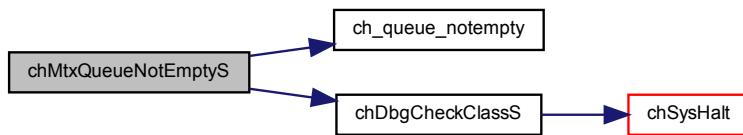
Returns

The mutex queue status.

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:

**7.27.4.11 chMtxGetOwnerI()**

```
static thread_t* chMtxGetOwnerI (
    mutex_t * mp ) [inline], [static]
```

Returns the mutex owner thread.

Parameters

out	<i>mp</i>	pointer to a <code>mutex_t</code> structure
-----	-----------	---

Returns

The owner thread.

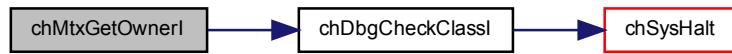
Return values

<code>NULL</code>	if the mutex is not owned.
-------------------	----------------------------

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:

**7.27.4.12 chMtxGetNextMutexX()**

```
static mutex_t* chMtxGetNextMutexX (
    void ) [inline], [static]
```

Returns the next mutex in the mutexes stack of the current thread.

Returns

A pointer to the next mutex in the stack.

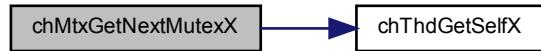
Return values

<code>NULL</code>	if the stack is empty.
-------------------	------------------------

Function Class:

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

Here is the call graph for this function:



7.28 Condition Variables

7.28.1 Detailed Description

This module implements the Condition Variables mechanism. Condition variables are an extensions to the mutex subsystem and cannot work alone.

Operation mode

The condition variable is a synchronization object meant to be used inside a zone protected by a mutex. Mutexes and condition variables together can implement a Monitor construct.

Precondition

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

Macros

- `#define __CONDVAR_DATA(name) {__CH_QUEUE_DATA(name.queue)}`
Data part of a static condition variable initializer.
- `#define CONDVAR_DECL(name) condition_variable_t name = __CONDVAR_DATA(name)`
Static condition variable initializer.

Typedefs

- `typedef struct condition_variable condition_variable_t`
condition_variable_t structure.

Data Structures

- `struct condition_variable`
condition_variable_t structure.

Functions

- `void chCondObjectInit (condition_variable_t *cp)`
Initializes a condition_variable_t structure.
- `void chCondSignal (condition_variable_t *cp)`
Signals one thread that is waiting on the condition variable.
- `void chCondSignall (condition_variable_t *cp)`
Signals one thread that is waiting on the condition variable.
- `void chCondBroadcast (condition_variable_t *cp)`
Signals all threads that are waiting on the condition variable.
- `void chCondBroadcastl (condition_variable_t *cp)`
Signals all threads that are waiting on the condition variable.
- `msg_t chCondWait (condition_variable_t *cp)`
Waits on the condition variable releasing the mutex lock.
- `msg_t chCondWaitS (condition_variable_t *cp)`
Waits on the condition variable releasing the mutex lock.
- `msg_t chCondWaitTimeout (condition_variable_t *cp, sysinterval_t timeout)`
Waits on the condition variable releasing the mutex lock.
- `msg_t chCondWaitTimeoutS (condition_variable_t *cp, sysinterval_t timeout)`
Waits on the condition variable releasing the mutex lock.

7.28.2 Macro Definition Documentation

7.28.2.1 __CONDVAR_DATA

```
#define __CONDVAR_DATA(
    name ) {__CH_QUEUE_DATA(name.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	<i>name</i>	the name of the condition variable
----	-------------	------------------------------------

7.28.2.2 CONDVAR_DECL

```
#define CONDVAR_DECL(
    name ) condition_variable_t name = __CONDVAR_DATA(name)
```

Static condition variable initializer.

Statically initialized condition variables require no explicit initialization using `chCondInit()`.

Parameters

in	<i>name</i>	the name of the condition variable
----	-------------	------------------------------------

7.28.3 Typedef Documentation

7.28.3.1 condition_variable_t

```
typedef struct condition_variable condition_variable_t
```

condition_variable_t structure.

7.28.4 Function Documentation

7.28.4.1 chCondObjectInit()

```
void chCondObjectInit (
    condition_variable_t * cp )
```

Initializes a `condition_variable_t` structure.

Parameters

out	<code>cp</code>	pointer to a <code>condition_variable_t</code> structure
-----	-----------------	--

Function Class:

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

Here is the call graph for this function:



7.28.4.2 chCondSignal()

```
void chCondSignal (
    condition_variable_t * cp )
```

Signals one thread that is waiting on the condition variable.

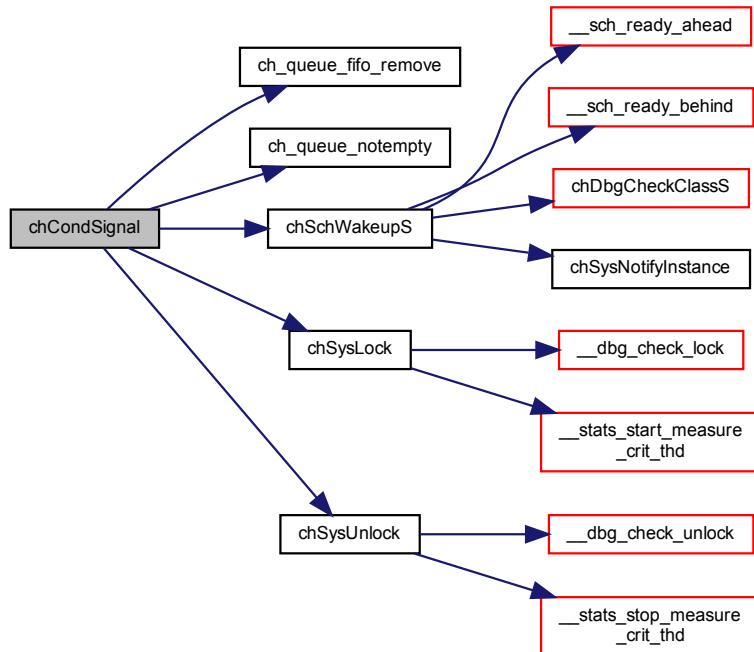
Parameters

in	<code>cp</code>	pointer to the <code>condition_variable_t</code> structure
----	-----------------	--

Function Class:

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

Here is the call graph for this function:

**7.28.4.3 chCondSignalI()**

```
void chCondSignalI (
    condition_variable_t * cp )
```

Signals one thread that is waiting on the condition variable.

Postcondition

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

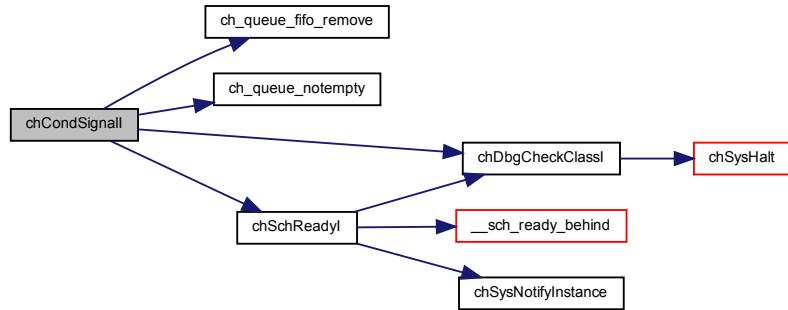
Parameters

in	<i>cp</i>	pointer to the condition_variable_t structure
----	-----------	---

Function Class:

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

Here is the call graph for this function:

7.28.4.4 `chCondBroadcast()`

```
void chCondBroadcast (
    condition_variable_t * cp )
```

Signals all threads that are waiting on the condition variable.

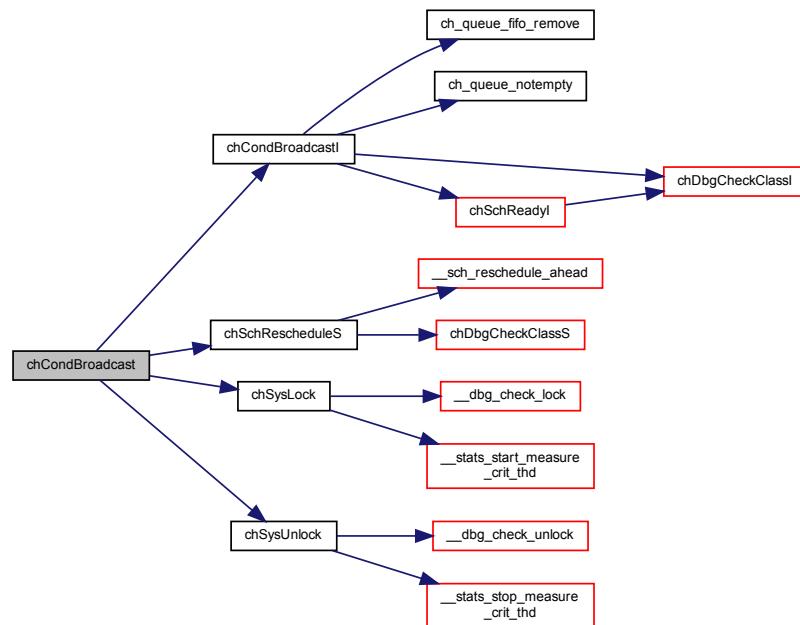
Parameters

in	<code>cp</code>	pointer to the <code>condition_variable_t</code> structure
----	-----------------	--

Function Class:

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

Here is the call graph for this function:

**7.28.4.5 chCondBroadcastI()**

```
void chCondBroadcastI (
    condition_variable_t * cp )
```

Signals all threads that are waiting on the condition variable.

Postcondition

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

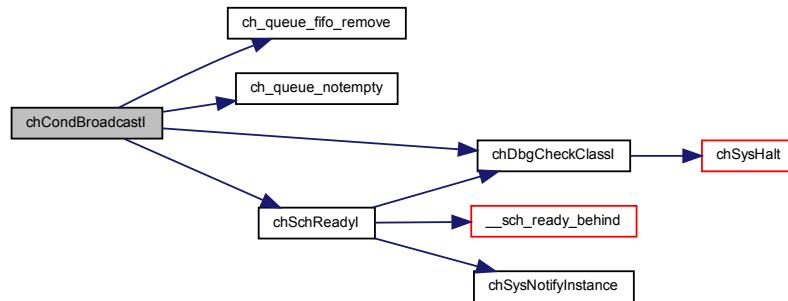
Parameters

in	<code>cp</code>	pointer to the <code>condition_variable_t</code> structure
----	-----------------	--

Function Class:

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

Here is the call graph for this function:



7.28.4.6 chCondWait()

```
msg_t chCondWait (
    condition_variable_t * cp )
```

Waits on the condition variable releasing the mutex lock.

Releases the currently owned mutex, waits on the condition variable, and finally acquires the mutex again. All the sequence is performed atomically.

Precondition

The invoking thread **must** have at least one owned mutex.

Parameters

in	<code>cp</code>	pointer to the <code>condition_variable_t</code> structure
----	-----------------	--

Returns

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

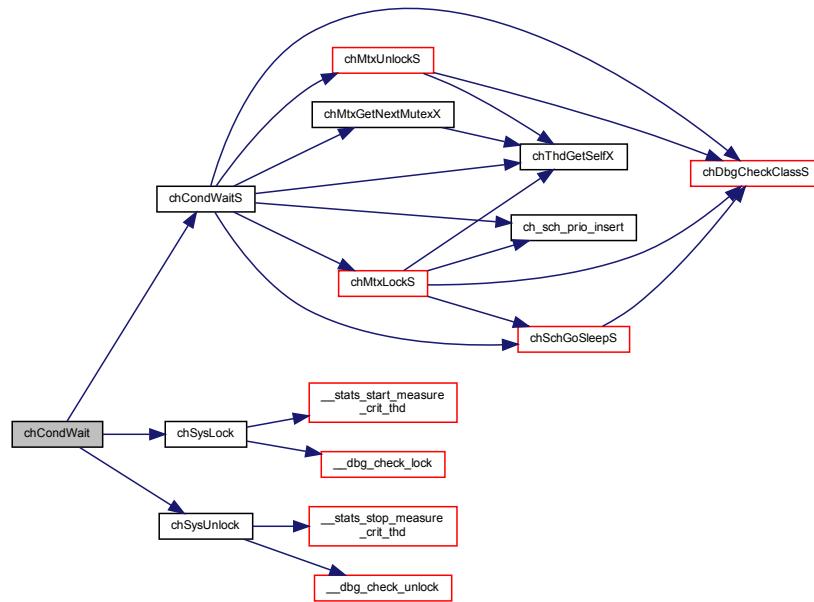
Return values

<code>MSG_OK</code>	if the condition variable has been signaled using <code>chCondSignal()</code> .
<code>MSG_RESET</code>	if the condition variable has been signaled using <code>chCondBroadcast()</code> .

Function Class:

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:

**7.28.4.7 chCondWaitS()**

```
msg_t chCondWaitS (
    condition_variable_t * cp )
```

Waits on the condition variable releasing the mutex lock.

Releases the currently owned mutex, waits on the condition variable, and finally acquires the mutex again. All the sequence is performed atomically.

Precondition

The invoking thread **must** have at least one owned mutex.

Parameters

in	<code>cp</code>	pointer to the <code>condition_variable_t</code> structure
----	-----------------	--

Returns

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

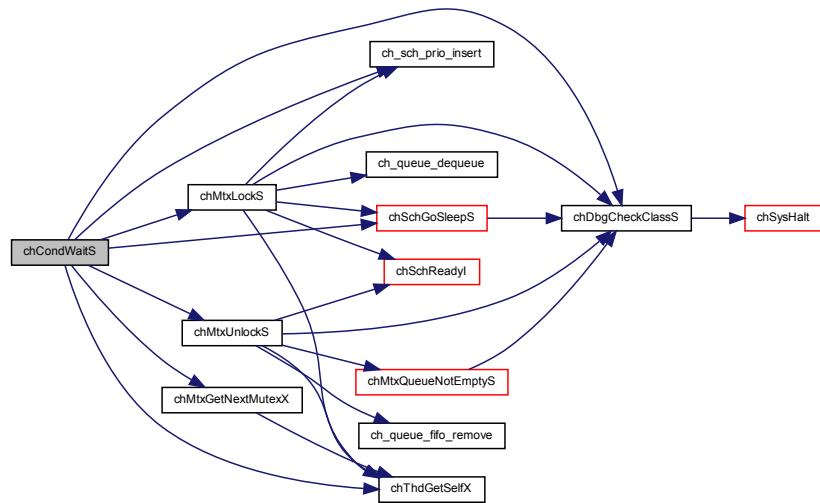
Return values

<code>MSG_OK</code>	if the condition variable has been signaled using <code>chCondSignal()</code> .
<code>MSG_RESET</code>	if the condition variable has been signaled using <code>chCondBroadcast()</code> .

Function Class:

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

Here is the call graph for this function:

**7.28.4.8 chCondWaitTimeout()**

```
msg_t chCondWaitTimeout (
    condition_variable_t * cp,
    sysinterval_t timeout )
```

Waits on the condition variable releasing the mutex lock.

Releases the currently owned mutex, waits on the condition variable, and finally acquires the mutex again. All the sequence is performed atomically.

Precondition

The invoking thread **must** have at least one owned mutex.

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

Postcondition

Exiting the function because a timeout does not re-acquire the mutex, the mutex ownership is lost.

Parameters

in	<i>cp</i>	pointer to the <code>condition_variable_t</code> structure
in	<i>timeout</i>	the number of ticks before the operation timeouts, the special values are handled as follow: <ul style="list-style-type: none"> • <code>TIME_INFINITE</code> no timeout. • <code>TIME_IMMEDIATE</code> this value is not allowed.

Returns

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

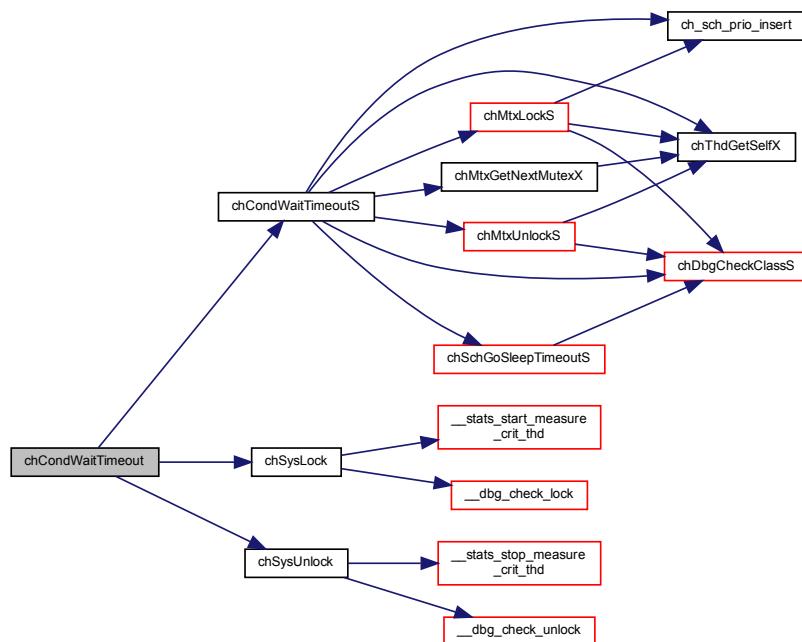
Return values

<code>MSG_OK</code>	if the condition variable has been signaled using <code>chCondSignal()</code> .
<code>MSG_RESET</code>	if the condition variable has been signaled using <code>chCondBroadcast()</code> .
<code>MSG_TIMEOUT</code>	if the condition variable has not been signaled within the specified timeout.

Function Class:

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

Here is the call graph for this function:



7.28.4.9 chCondWaitTimeoutS()

```
msg_t chCondWaitTimeoutS (
    condition_variable_t * cp,
    sysinterval_t timeout )
```

Waits on the condition variable releasing the mutex lock.

Releases the currently owned mutex, waits on the condition variable, and finally acquires the mutex again. All the sequence is performed atomically.

Precondition

The invoking thread **must** have at least one owned mutex.

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

Postcondition

Exiting the function because a timeout does not re-acquire the mutex, the mutex ownership is lost.

Parameters

in	<i>cp</i>	pointer to the <code>condition_variable_t</code> structure
in	<i>timeout</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> no timeout. • <i>TIME_IMMEDIATE</i> this value is not allowed.

Returns

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

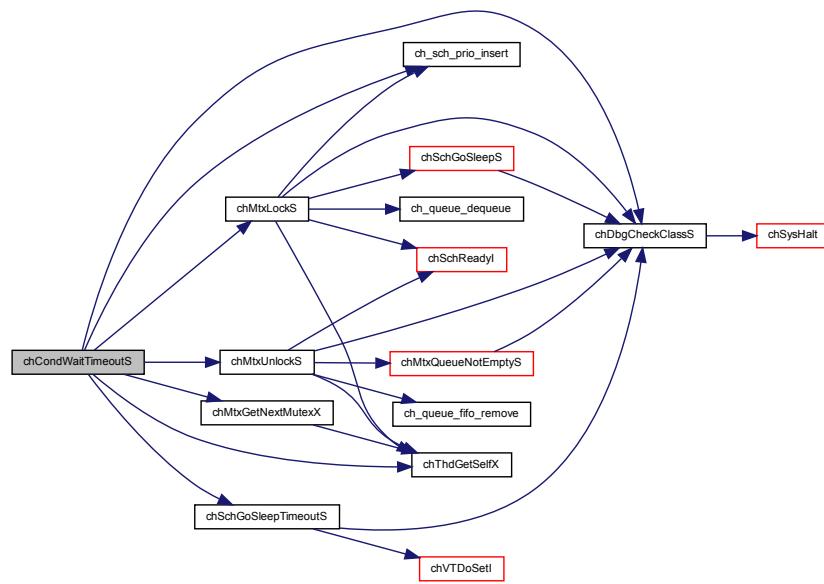
Return values

<i>MSG_OK</i>	if the condition variable has been signaled using <code>chCondSignal()</code> .
<i>MSG_RESET</i>	if the condition variable has been signaled using <code>chCondBroadcast()</code> .
<i>MSG_TIMEOUT</i>	if the condition variable has not been signaled within the specified timeout.

Function Class:

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

Here is the call graph for this function:



7.29 Event Flags

7.29.1 Detailed Description

Event Flags, Event Sources and Event Listeners.

Operation mode

Each thread has a mask of pending events inside its `thread_t` structure. Operations defined for events:

- **Wait**, the invoking thread goes to sleep until a certain AND/OR combination of events are signaled.
- **Clear**, a mask of events is cleared from the pending events, the cleared events mask is returned (only the events that were actually pending and then cleared).
- **Signal**, an events mask is directly ORed to the mask of the signaled thread.
- **Broadcast**, each thread registered on an Event Source is signaled with the events specified in its Event Listener.
- **Dispatch**, an events mask is scanned and for each bit set to one an associated handler function is invoked. Bit masks are scanned from bit zero upward.

An Event Source is a special object that can be "broadcasted" by a thread or an interrupt service routine. Broadcasting an Event Source has the effect that all the threads registered on the Event Source will be signaled with an events mask.

An unlimited number of Event Sources can exists in a system and each thread can be listening on an unlimited number of them.

Precondition

In order to use the Events APIs the `CH_CFG_USE_EVENTS` option must be enabled in `chconf.h`.

Postcondition

Enabling events requires 1-4 (depending on the architecture) extra bytes in the `thread_t` structure.

Macros

- `#define ALL_EVENTS ((eventmask_t)-1)`
All events allowed mask.
- `#define EVENT_MASK(eid) ((eventmask_t)1 << (eventmask_t)(eid))`
Returns an event mask from an event identifier.
- `#define __EVENTSOURCE_DATA(name) { (event_listener_t *)(&name)}`
Data part of a static event source initializer.
- `#define EVENTSOURCE_DECL(name) event_source_t name = __EVENTSOURCE_DATA(name)`
Static event source initializer.

Typedefs

- `typedef struct event_source event_source_t`
Event Source structure.
- `typedef void(* evhandler_t) (eventid_t id)`
Event Handler callback function.

Data Structures

- `struct event_listener`
Event Listener structure.
- `struct event_source`
Event Source structure.

Functions

- `void chEvtRegisterMaskWithFlagsI (event_source_t *esp, event_listener_t *elp, eventmask_t events, eventflags_t wflags)`
Registers an Event Listener on an Event Source.
- `void chEvtRegisterMaskWithFlags (event_source_t *esp, event_listener_t *elp, eventmask_t events, eventflags_t wflags)`
Registers an Event Listener on an Event Source.
- `void chEvtUnregister (event_source_t *esp, event_listener_t *elp)`
Unregisters an Event Listener from its Event Source.
- `eventmask_t chEvtGetAndClearEventsI (eventmask_t events)`
Clears the pending events specified in the events mask.
- `eventmask_t chEvtGetAndClearEvents (eventmask_t events)`
Clears the pending events specified in the events mask.
- `eventmask_t chEvtAddEvents (eventmask_t events)`
*Adds (OR) a set of events to the current thread, this is **much** faster than using `chEvtBroadcast()` or `chEvtSignal()`.*
- `eventflags_t chEvtGetAndClearFlagsI (event_listener_t *elp)`
Returns the unmasked flags associated to an `event_listener_t`.
- `eventflags_t chEvtGetAndClearFlags (event_listener_t *elp)`
Returns the flags associated to an `event_listener_t`.
- `void chEvtSignall (thread_t *tp, eventmask_t events)`
Adds a set of event flags directly to the specified `thread_t`.
- `void chEvtSignal (thread_t *tp, eventmask_t events)`
Adds a set of event flags directly to the specified `thread_t`.
- `void chEvtBroadcastFlagsI (event_source_t *esp, eventflags_t flags)`
Signals all the Event Listeners registered on the specified Event Source.
- `void chEvtBroadcastFlags (event_source_t *esp, eventflags_t flags)`
Signals all the Event Listeners registered on the specified Event Source.
- `void chEvtDispatch (const evhandler_t *handlers, eventmask_t events)`
Invokes the event handlers associated to an event flags mask.
- `eventmask_t chEvtWaitOne (eventmask_t events)`
Waits for exactly one of the specified events.
- `eventmask_t chEvtWaitAny (eventmask_t events)`
Waits for any of the specified events.
- `eventmask_t chEvtWaitAll (eventmask_t events)`

- `eventmask_t chEvtWaitOneTimeout (eventmask_t events, sysinterval_t timeout)`
Waits for exactly one of the specified events.
- `eventmask_t chEvtWaitAnyTimeout (eventmask_t events, sysinterval_t timeout)`
Waits for any of the specified events.
- `eventmask_t chEvtWaitAllTimeout (eventmask_t events, sysinterval_t timeout)`
Waits for all the specified events.
- `static void chEvtObjectInit (event_source_t *esp)`
Initializes an Event Source.
- `static void chEvtRegisterMask (event_source_t *esp, event_listener_t *elp, eventmask_t events)`
Registers an Event Listener on an Event Source.
- `static void chEvtRegister (event_source_t *esp, event_listener_t *elp, eventid_t event)`
Registers an Event Listener on an Event Source.
- `static bool chEvtIsListeningI (event_source_t *esp)`
Verifies if there is at least one event_listener_t registered.
- `static void chEvtBroadcast (event_source_t *esp)`
Signals all the Event Listeners registered on the specified Event Source.
- `static void chEvtBroadcastI (event_source_t *esp)`
Signals all the Event Listeners registered on the specified Event Source.
- `static eventmask_t chEvtAddEventsI (eventmask_t events)`
*Adds (OR) a set of events to the current thread, this is **much** faster than using `chEvtBroadcast ()` or `chEvtSignal ()`.*
- `static eventmask_t chEvtGetEventsX (void)`
Returns the events mask.

7.29.2 Macro Definition Documentation

7.29.2.1 ALL_EVENTS

```
#define ALL_EVENTS ((eventmask_t)-1)
```

All events allowed mask.

7.29.2.2 EVENT_MASK

```
#define EVENT_MASK(
    eid ) ((eventmask_t)1 << (eventmask_t)(eid))
```

Returns an event mask from an event identifier.

7.29.2.3 __EVENTSOURCE_DATA

```
#define __EVENTSOURCE_DATA (
    name ) { (event_listener_t *)(&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

<i>name</i>	the name of the event source variable
-------------	---------------------------------------

7.29.2.4 EVENTSOURCE_DECL

```
#define EVENTSOURCE_DECL( name ) event_source_t name = __EVENTSOURCE_DATA(name)
```

Static event source initializer.

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

Parameters

<i>name</i>	the name of the event source variable
-------------	---------------------------------------

7.29.3 Typedef Documentation**7.29.3.1 event_source_t**

```
typedef struct event_source event_source_t
```

Event Source structure.

7.29.3.2 evhandler_t

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

Event Handler callback function.

7.29.4 Function Documentation

7.29.4.1 chEvtRegisterMaskWithFlagsI()

```
void chEvtRegisterMaskWithFlagsI (
    event_source_t * esp,
    event_listener_t * elp,
    eventmask_t events,
    eventflags_t wflags )
```

Registers an Event Listener on an Event Source.

Once a thread has registered as listener on an event source it will be notified of all events broadcasted there.

Note

Multiple Event Listeners can specify the same bits to be ORed to different threads.

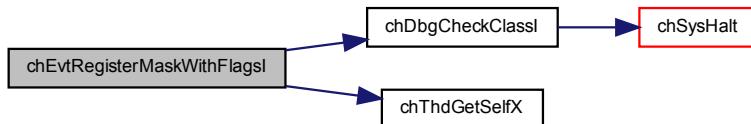
Parameters

in	<i>esp</i>	pointer to the <code>event_source_t</code> structure
in	<i>elp</i>	pointer to the <code>event_listener_t</code> structure
in	<i>events</i>	events to be ORed to the thread when the event source is broadcasted
in	<i>wflags</i>	mask of flags the listening thread is interested in

Function Class:

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:



7.29.4.2 chEvtRegisterMaskWithFlags()

```
void chEvtRegisterMaskWithFlags (
    event_source_t * esp,
    event_listener_t * elp,
    eventmask_t events,
    eventflags_t wflags )
```

Registers an Event Listener on an Event Source.

Once a thread has registered as listener on an event source it will be notified of all events broadcasted there.

Note

Multiple Event Listeners can specify the same bits to be ORed to different threads.

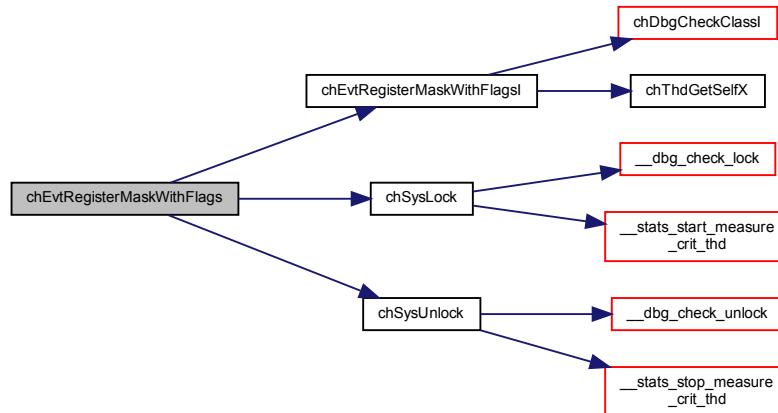
Parameters

in	<i>esp</i>	pointer to the <code>event_source_t</code> structure
in	<i>elp</i>	pointer to the <code>event_listener_t</code> structure
in	<i>events</i>	events to be ORed to the thread when the event source is broadcasted
in	<i>wflags</i>	mask of flags the listening thread is interested in

Function Class:

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

Here is the call graph for this function:

**7.29.4.3 chEvtUnregister()**

```
void chEvtUnregister (
    event_source_t * esp,
    event_listener_t * elp )
```

Unregisters an Event Listener from its Event Source.

Note

If the event listener is not registered on the specified event source then the function does nothing.

For optimal performance it is better to perform the unregister operations in inverse order of the register operations (elements are found on top of the list).

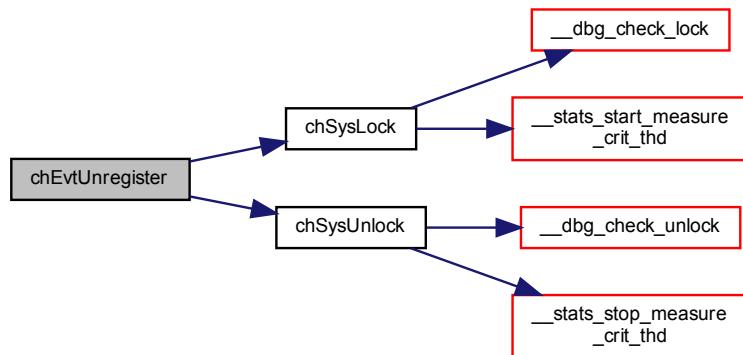
Parameters

in	<i>esp</i>	pointer to the <code>event_source_t</code> structure
in	<i>elp</i>	pointer to the <code>event_listener_t</code> structure

Function Class:

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

Here is the call graph for this function:



7.29.4.4 chEvtGetAndClearEventsI()

```
eventmask_t chEvtGetAndClearEventsI (
    eventmask_t events )
```

Clears the pending events specified in the events mask.

Parameters

in	<i>events</i>	the events to be cleared
----	---------------	--------------------------

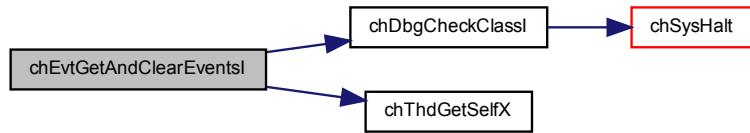
Returns

The mask of pending events that were cleared.

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:



7.29.4.5 chEvtGetAndClearEvents()

```
eventmask_t chEvtGetAndClearEvents (
    eventmask_t events )
```

Clears the pending events specified in the events mask.

Parameters

in	events	the events to be cleared
----	--------	--------------------------

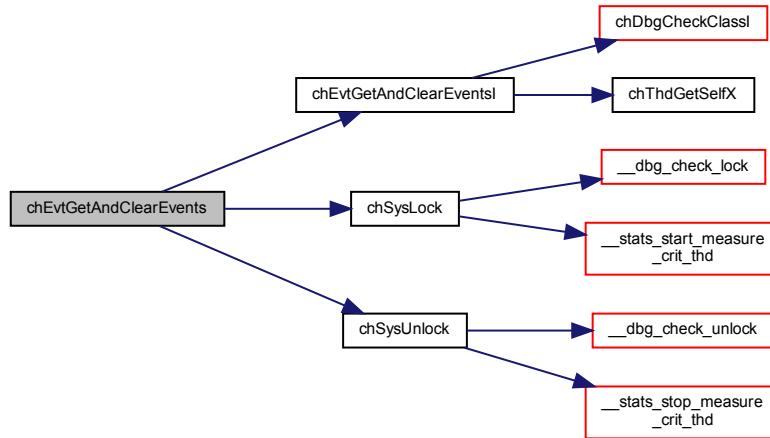
Returns

The mask of 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:



7.29.4.6 chEvtAddEvents()

```
eventmask_t chEvtAddEvents (
    eventmask_t events )
```

Adds (OR) a set of events to the current thread, this is **much** faster than using `chEvtBroadcast()` or `chEvtSignal()`.

Parameters

in	<code>events</code>	the events to be added
----	---------------------	------------------------

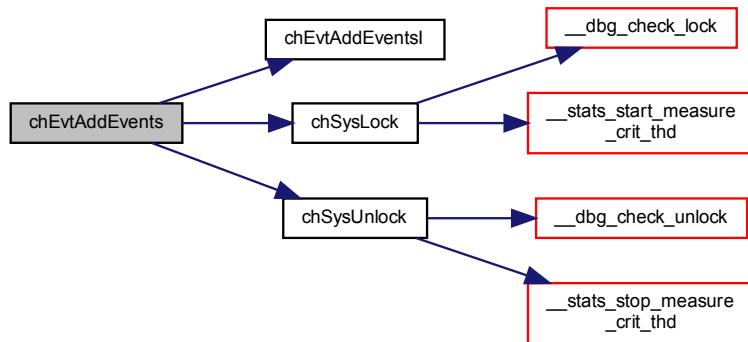
Returns

The mask of currently pending events.

Function Class:

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

Here is the call graph for this function:

**7.29.4.7 chEvtGetAndClearFlagsI()**

```
eventflags_t chEvtGetAndClearFlagsI (
    event_listener_t * elp )
```

Returns the unmasked flags associated to an `event_listener_t`.

The flags are returned and the `event_listener_t` flags mask is cleared.

Parameters

in	<code>elp</code>	pointer to the <code>event_listener_t</code> structure
----	------------------	--

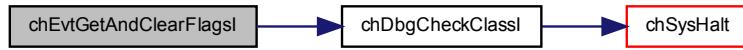
Returns

The flags added to the listener by the associated event source.

Function Class:

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

Here is the call graph for this function:



7.29.4.8 chEvtGetAndClearFlags()

```
eventflags_t chEvtGetAndClearFlags (
    event_listener_t * elp )
```

Returns the flags associated to an `event_listener_t`.

The flags are returned and the `event_listener_t` flags mask is cleared.

Parameters

in	<code>elp</code>	pointer to the <code>event_listener_t</code> structure
----	------------------	--

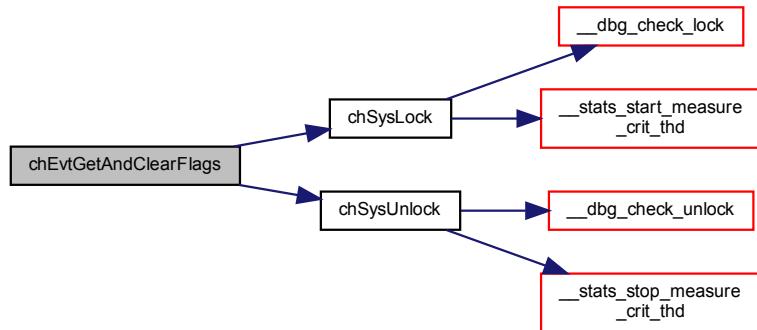
Returns

The flags added to the listener by the associated event source.

Function Class:

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

Here is the call graph for this function:

**7.29.4.9 chEvtSignalI()**

```

void chEvtSignalI (
    thread_t * tp,
    eventmask_t events )
  
```

Adds a set of event flags directly to the specified `thread_t`.

Postcondition

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

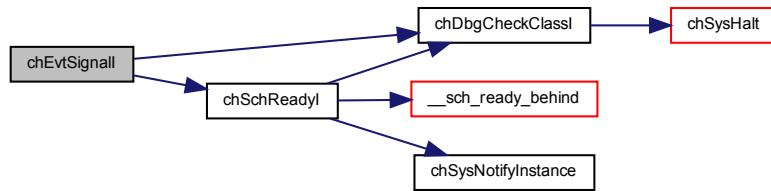
Parameters

in	<code>tp</code>	the thread to be signaled
in	<code>events</code>	the events set to be ORed

Function Class:

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

Here is the call graph for this function:

**7.29.4.10 chEvtSignal()**

```

void chEvtSignal (
    thread_t * tp,
    eventmask_t events )
  
```

Adds a set of event flags directly to the specified `thread_t`.

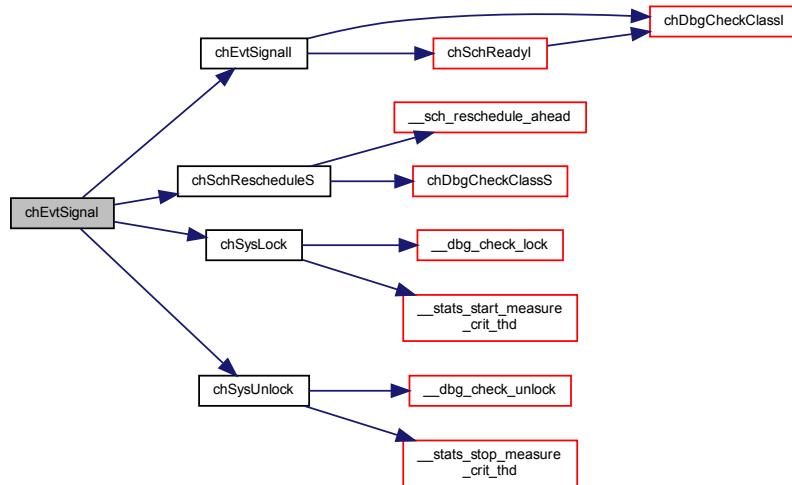
Parameters

in	<code>tp</code>	the thread to be signaled
in	<code>events</code>	the events set to be ORed

Function Class:

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

Here is the call graph for this function:



7.29.4.11 chEvtBroadcastFlagsI()

```

void chEvtBroadcastFlagsI (
    event_source_t * esp,
    eventflags_t flags )
  
```

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

This function variants ORs the specified event flags to all the threads registered on the `event_source_t` in addition to the event flags specified by the threads themselves in the `event_listener_t` objects.

Postcondition

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

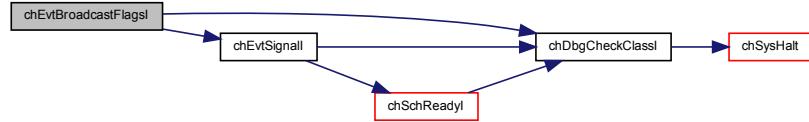
Parameters

in	<code>esp</code>	pointer to the <code>event_source_t</code> structure
in	<code>flags</code>	the flags set to be added to the listener flags mask

Function Class:

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

Here is the call graph for this function:



7.29.4.12 chEvtBroadcastFlags()

```

void chEvtBroadcastFlags (
    event_source_t * esp,
    eventflags_t flags )
  
```

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

This function variants ORs the specified event flags to all the threads registered on the `event_source_t` in addition to the event flags specified by the threads themselves in the `event_listener_t` objects.

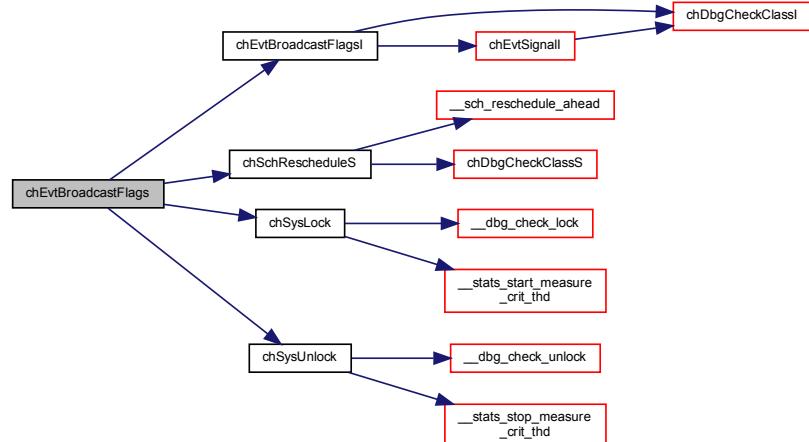
Parameters

in	<code>esp</code>	pointer to the <code>event_source_t</code> structure
in	<code>flags</code>	the flags set to be added to the listener flags mask

Function Class:

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

Here is the call graph for this function:



7.29.4.13 chEvtDispatch()

```
void chEvtDispatch (
    const evhandler_t * handlers,
    eventmask_t events )
```

Invokes the event handlers associated to an event flags mask.

Parameters

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

Function Class:

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

7.29.4.14 chEvtWaitOne()

```
eventmask_t chEvtWaitOne (
    eventmask_t events )
```

Waits for exactly one of the specified events.

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

Note

One and only one event is served in the function, the one with the lowest event id. The function is meant to be invoked into a loop in order to serve all the pending events.

This means that Event Listeners with a lower event identifier have an higher priority.

Parameters

in	<i>events</i>	events that the function should wait for, <code>ALL_EVENTS</code> enables all the events
----	---------------	--

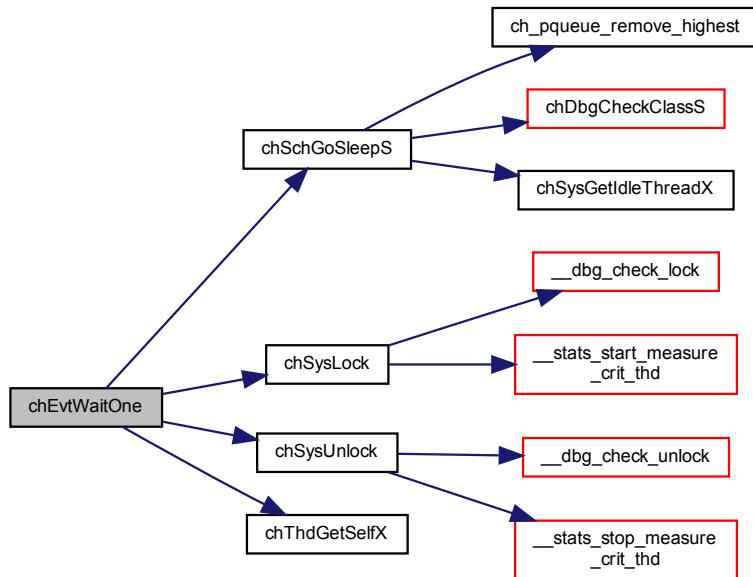
Returns

The mask of the lowest event id served and cleared.

Function Class:

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

Here is the call graph for this function:



7.29.4.15 chEvtWaitAny()

```
eventmask_t chEvtWaitAny (
    eventmask_t events )
```

Waits for any of the specified events.

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

Parameters

in	<code>events</code>	events that the function should wait for, <code>ALL_EVENTS</code> enables all the events
----	---------------------	--

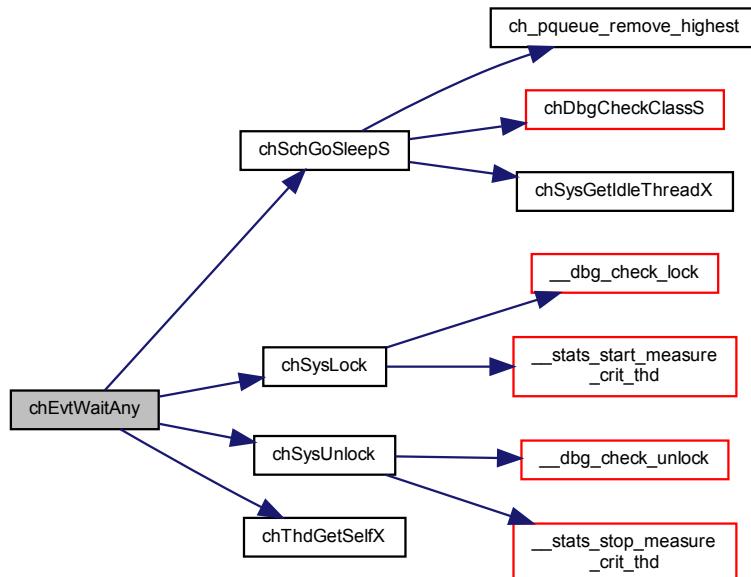
Returns

The mask of the served and cleared events.

Function Class:

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

Here is the call graph for this function:

**7.29.4.16 chEvtWaitAll()**

```
eventmask_t chEvtWaitAll (
    eventmask_t events )
```

Waits for all the specified events.

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

Parameters

in	<code>events</code>	events that the function should wait for, ALL_EVENTS requires all the events
----	---------------------	--

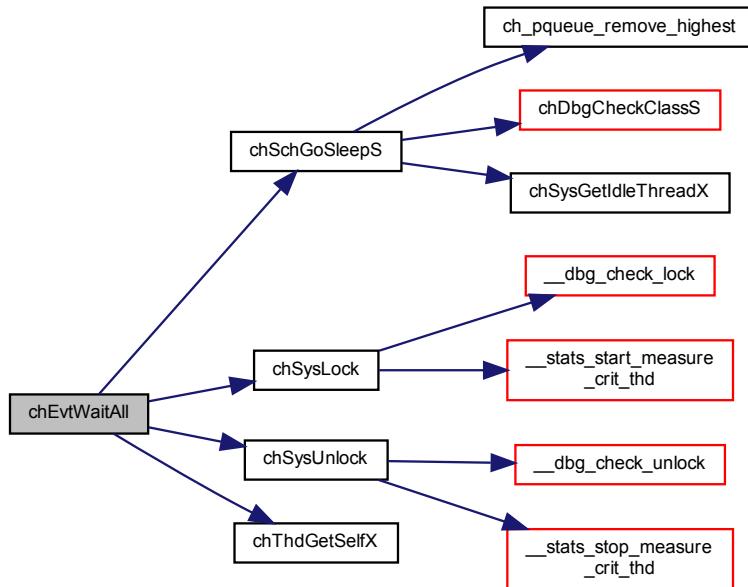
Returns

The mask of the served and cleared events.

Function Class:

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

Here is the call graph for this function:



7.29.4.17 chEvtWaitOneTimeout()

```
eventmask_t chEvtWaitOneTimeout (
    eventmask_t events,
    sysinterval_t timeout )
```

Waits for exactly one of the specified events.

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

Note

One and only one event is served in the function, the one with the lowest event id. The function is meant to be invoked into a loop in order to serve all the pending events.

This means that Event Listeners with a lower event identifier have an higher priority.

Parameters

in	<code>events</code>	events that the function should wait for, <code>ALL_EVENTS</code> enables all the events
in	<code>timeout</code>	the number of ticks before the operation timeouts, the following special values are allowed:
Chibios/RT		<ul style="list-style-type: none"> • <code>TIME_IMMEDIATE</code> immediate timeout. • <code>TIME_INFINITE</code> no timeout.

Returns

The mask of the lowest event id served and cleared.

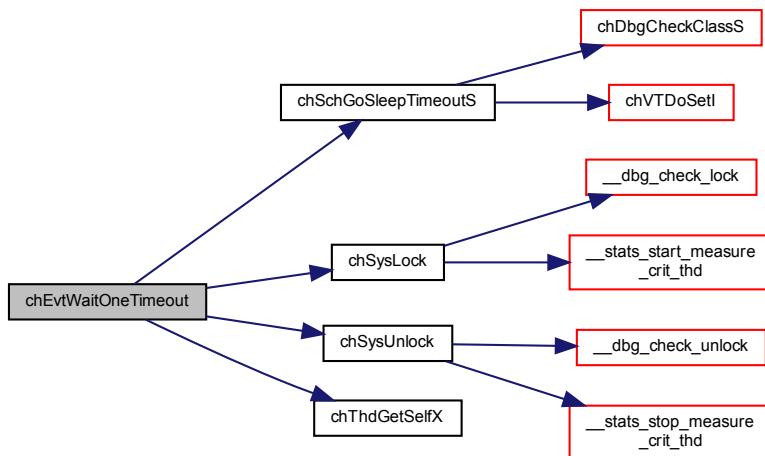
Return values

<i>0</i>	if the operation has timed out.
----------	---------------------------------

Function Class:

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

Here is the call graph for this function:

**7.29.4.18 chEvtWaitAnyTimeout()**

```
eventmask_t chEvtWaitAnyTimeout (
    eventmask_t events,
    sysinterval_t timeout )
```

Waits for any of the specified events.

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

Parameters

in	<i>events</i>	events that the function should wait for, <code>ALL_EVENTS</code> enables all the events
in	<i>timeout</i>	the number of ticks before the operation timeouts, the following special values are allowed: <ul style="list-style-type: none"> • <code>TIME_IMMEDIATE</code> immediate timeout. • <code>TIME_INFINITE</code> no timeout.
		ChibiOS/RT

Returns

The mask of the served and cleared events.

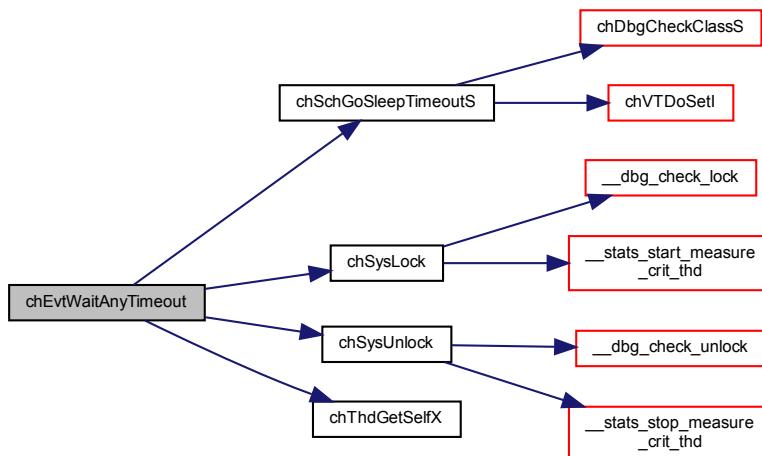
Return values

<i>0</i>	if the operation has timed out.
----------	---------------------------------

Function Class:

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

Here is the call graph for this function:

**7.29.4.19 chEvtWaitAllTimeout()**

```
eventmask_t chEvtWaitAllTimeout (
    eventmask_t events,
    sysinterval_t timeout )
```

Waits for all the specified events.

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

Parameters

in	<code>events</code>	events that the function should wait for, <code>ALL_EVENTS</code> requires all the events
in	<code>timeout</code>	the number of ticks before the operation timeouts, the following special values are allowed: <ul style="list-style-type: none"> • <code>TIME_IMMEDIATE</code> immediate timeout.
ChibiOS/RT		<ul style="list-style-type: none"> • <code>TIME_INFINITE</code> no timeout.

Returns

The mask of the served and cleared events.

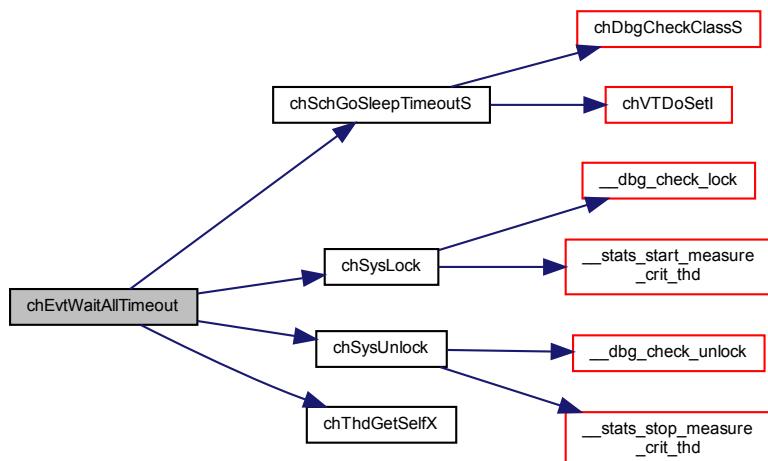
Return values

<code>0</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:

**7.29.4.20 chEvtObjectInit()**

```
static void chEvtObjectInit (
    event_source_t * esp ) [inline], [static]
```

Initializes an Event Source.

Note

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

Parameters

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

Function Class:

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

7.29.4.21 chEvtRegisterMask()

```
static void chEvtRegisterMask (
    event_source_t * esp,
    event_listener_t * elp,
    eventmask_t events ) [inline], [static]
```

Registers an Event Listener on an Event Source.

Once a thread has registered as listener on an event source it will be notified of all events broadcasted there.

Note

Multiple Event Listeners can specify the same bits to be ORed to different threads.

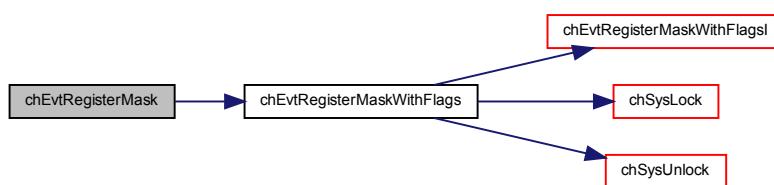
Parameters

in	<i>esp</i>	pointer to the <code>event_source_t</code> structure
out	<i>elp</i>	pointer to the <code>event_listener_t</code> structure
in	<i>events</i>	the mask of events to be ORed to the thread when the event source is broadcasted

Function Class:

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

Here is the call graph for this function:

**7.29.4.22 chEvtRegister()**

```
static void chEvtRegister (
    event_source_t * esp,
```

```
event_listener_t * elp,
eventid_t event ) [inline], [static]
```

Registers an Event Listener on an Event Source.

Note

Multiple Event Listeners can use the same event identifier, the listener will share the callback function.

Parameters

in	<code>esp</code>	pointer to the <code>event_source_t</code> structure
out	<code>elp</code>	pointer to the <code>event_listener_t</code> structure
in	<code>event</code>	numeric identifier assigned to the Event Listener. The value must range between zero and the size, in bit, of the <code>eventmask_t</code> type minus one.

Function Class:

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

Here is the call graph for this function:



7.29.4.23 chEvtIsListeningI()

```
static bool chEvtIsListeningI (
    event_source_t * esp ) [inline], [static]
```

Verifies if there is at least one `event_listener_t` registered.

Parameters

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

Returns

The event source status.

Function Class:

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

7.29.4.24 chEvtBroadcast()

```
static void chEvtBroadcast (
    event_source_t * esp ) [inline], [static]
```

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

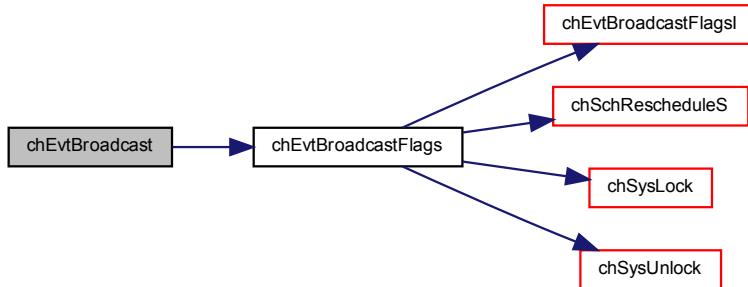
Parameters

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

Function Class:

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

Here is the call graph for this function:

**7.29.4.25 chEvtBroadcastI()**

```
static void chEvtBroadcastI (
    event_source_t * esp ) [inline], [static]
```

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

Postcondition

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

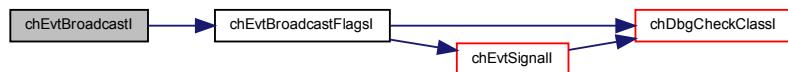
Parameters

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

Function Class:

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

Here is the call graph for this function:

**7.29.4.26 chEvtAddEventsI()**

```
static eventmask_t chEvtAddEventsI (
    eventmask_t events ) [inline], [static]
```

Adds (OR) a set of events to the current thread, this is **much** faster than using `chEvtBroadcast()` or `chEvtSignal()`.

Parameters

in	<code>events</code>	the events to be added
----	---------------------	------------------------

Returns

The mask of currently pending events.

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.

7.29.4.27 chEvtGetEventsX()

```
static eventmask_t chEvtGetEventsX (
    void ) [inline], [static]
```

Returns the events mask.

The pending events mask is returned but not altered in any way.

Returns

The pending events mask.

Function Class:

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

7.30 Synchronous Messages

7.30.1 Detailed Description

Synchronous inter-thread messages APIs and services.

Operation Mode

Synchronous messages are an easy to use and fast IPC mechanism, threads can both act as message servers and/or message clients, the mechanism allows data to be carried in both directions. Note that messages are not copied between the client and server threads but just a pointer passed so the exchange is very time efficient.

Messages are scalar data types of type `msg_t` that are guaranteed to be size compatible with data pointers. Note that on some architectures function pointers can be larger than `msg_t`.

Messages are usually processed in FIFO order but it is possible to process them in priority order by enabling the `CH_CFG_USE_MESSAGES_PRIORITY` option in `chconf.h`.

Precondition

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

Postcondition

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

Functions

- `msg_t chMsgSend (thread_t *tp, msg_t msg)`
Sends a message to the specified thread.
- `thread_t * chMsgWaitS (void)`
Suspends the thread and waits for an incoming message.
- `thread_t * chMsgWaitTimeoutS (sysinterval_t timeout)`
Suspends the thread and waits for an incoming message or a timeout to occur.
- `thread_t * chMsgPollS (void)`
Poll to check for an incoming message.
- `void chMsgRelease (thread_t *tp, msg_t msg)`
Releases a sender thread specifying a response message.
- `static thread_t * chMsgWait (void)`
Suspends the thread and waits for an incoming message.
- `static thread_t * chMsgWaitTimeout (sysinterval_t timeout)`
Suspends the thread and waits for an incoming message or a timeout to occur.
- `static thread_t * chMsgPoll (void)`
Poll to check for an incoming message.
- `static bool chMsgIsPendingI (thread_t *tp)`
Evaluates to true if the thread has pending messages.
- `static msg_t chMsgGet (thread_t *tp)`
Returns the message carried by the specified thread.
- `static void chMsgReleaseS (thread_t *tp, msg_t msg)`
Releases the thread waiting on top of the messages queue.

7.30.2 Function Documentation

7.30.2.1 chMsgSend()

```
msg_t chMsgSend (
    thread_t * tp,
    msg_t msg )
```

Sends a message to the specified thread.

The sender is stopped until the receiver executes a [chMsgRelease \(\)](#) after receiving the message.

Parameters

in	<i>tp</i>	the pointer to the thread
in	<i>msg</i>	the message

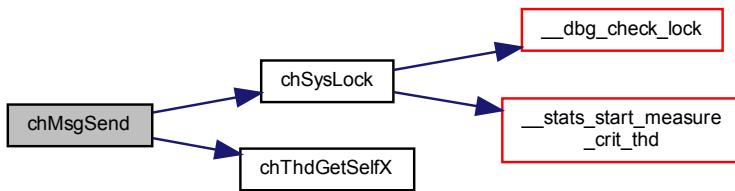
Returns

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:



7.30.2.2 chMsgWaitS()

```
thread_t * chMsgWaitS (
    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.

The reference counter of the sender thread is not increased, the returned pointer is a temporary reference.

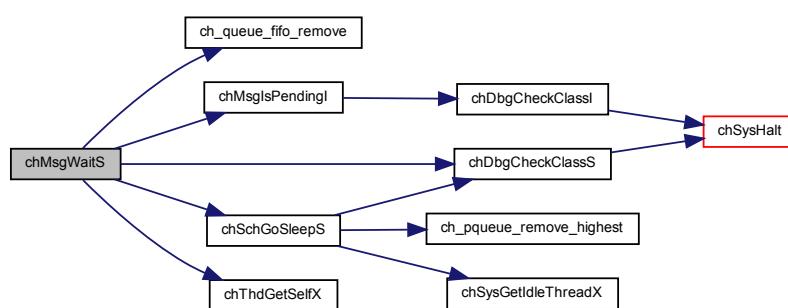
Returns

A pointer to the thread carrying the 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:

**7.30.2.3 chMsgWaitTimeoutS()**

```
thread_t * chMsgWaitTimeoutS (
    sysinterval_t timeout )
```

Suspends the thread and waits for an incoming message or a timeout to occur.

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.

The reference counter of the sender thread is not increased, the returned pointer is a temporary reference.

Parameters

in	<i>timeout</i>	the number of ticks before the operation timeouts, the following special values are allowed:
		• <i>TIME_INFINITE</i> no timeout.

Returns

A pointer to the thread carrying the message.

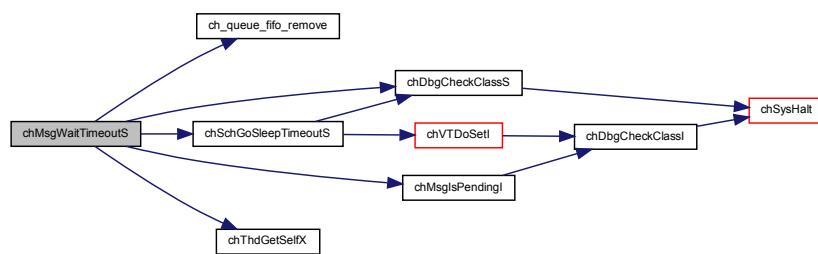
Return values

<i>NULL</i>	if a timeout occurred.
-------------	------------------------

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:

**7.30.2.4 chMsgPollS()**

```
thread_t * chMsgPollS (
    void )
```

Poll to check for an incoming message.

Postcondition

If a message is available 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.

The reference counter of the sender thread is not increased, the returned pointer is a temporary reference.

Returns

Result of the poll.

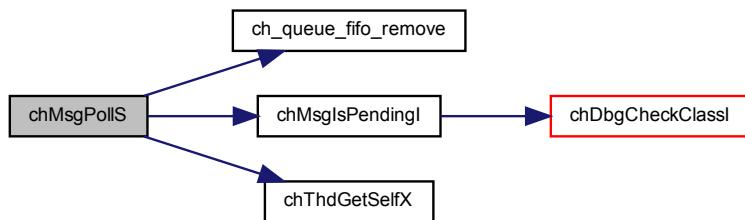
Return values

<code>NULL</code>	if no incoming message waiting.
-------------------	---------------------------------

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:



7.30.2.5 `chMsgRelease()`

```
void chMsgRelease (
    thread_t * tp,
    msg_t msg )
```

Releases a sender thread specifying a response message.

Precondition

Invoke this function only after a message has been received using `chMsgWait()`.

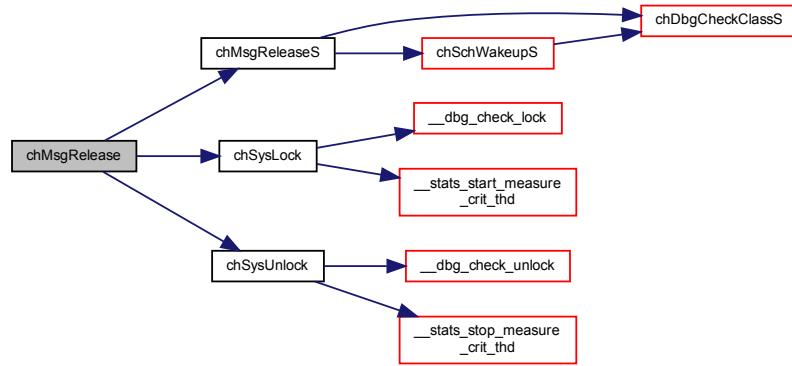
Parameters

in	<i>tp</i>	pointer to the thread
in	<i>msg</i>	message to be returned to the sender

Function Class:

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

Here is the call graph for this function:

**7.30.2.6 chMsgWait()**

```
static thread_t* chMsgWait (
    void ) [inline], [static]
```

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.

The reference counter of the sender thread is not increased, the returned pointer is a temporary reference.

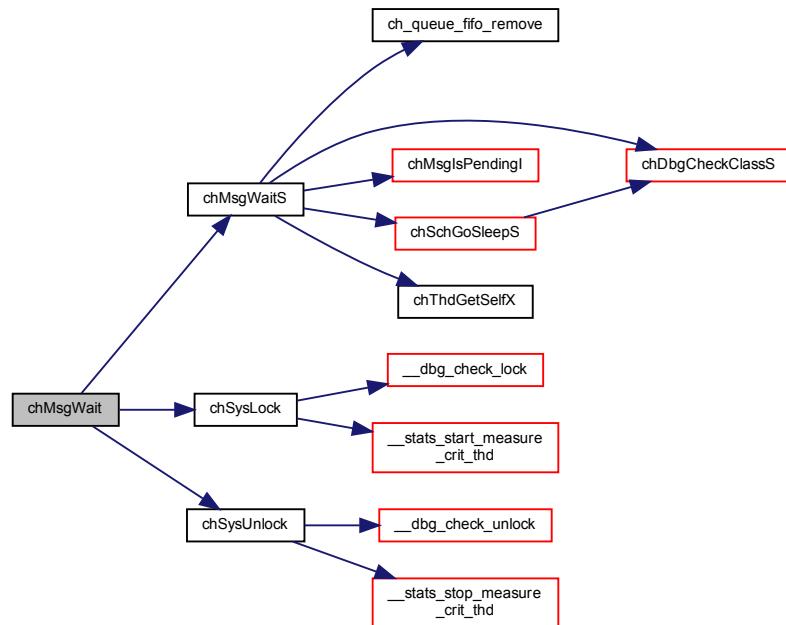
Returns

A pointer 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:



7.30.2.7 chMsgWaitTimeout()

```
static thread_t* chMsgWaitTimeout (
    sysinterval_t timeout ) [inline], [static]
```

Suspends the thread and waits for an incoming message or a timeout to occur.

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.

The reference counter of the sender thread is not increased, the returned pointer is a temporary reference.

Parameters

in	<i>timeout</i>	the number of ticks before the operation timeouts, the following special values are allowed:
		<ul style="list-style-type: none"> • <i>TIME_IMMEDIATE</i> immediate timeout.
		<ul style="list-style-type: none"> • <i>TIME_INFINITE</i> no timeout.

Returns

A pointer to the thread carrying the message.

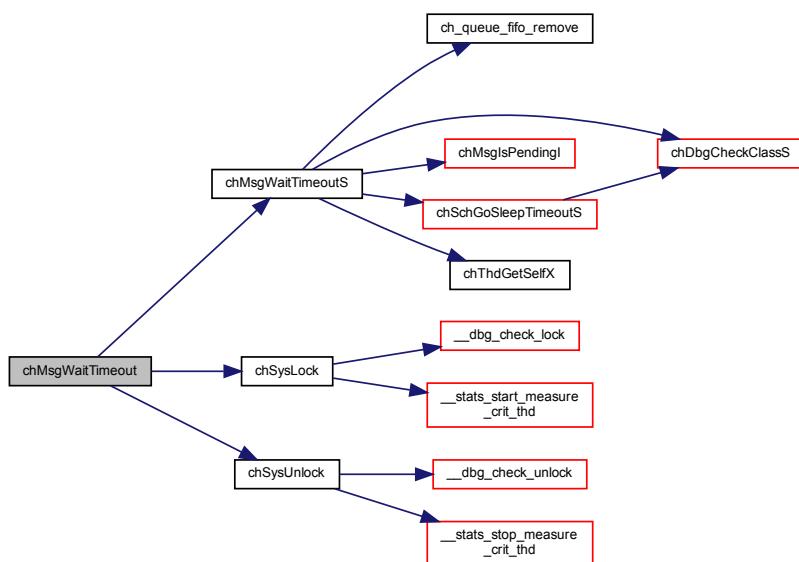
Return values

<i>NULL</i>	if a timeout occurred.
-------------	------------------------

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:



7.30.2.8 chMsgPoll()

```
static thread_t* chMsgPoll (
    void ) [inline], [static]
```

Poll to check for an incoming message.

Postcondition

If a message is available 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.

The reference counter of the sender thread is not increased, the returned pointer is a temporary reference.

Returns

A pointer to the thread carrying the message.

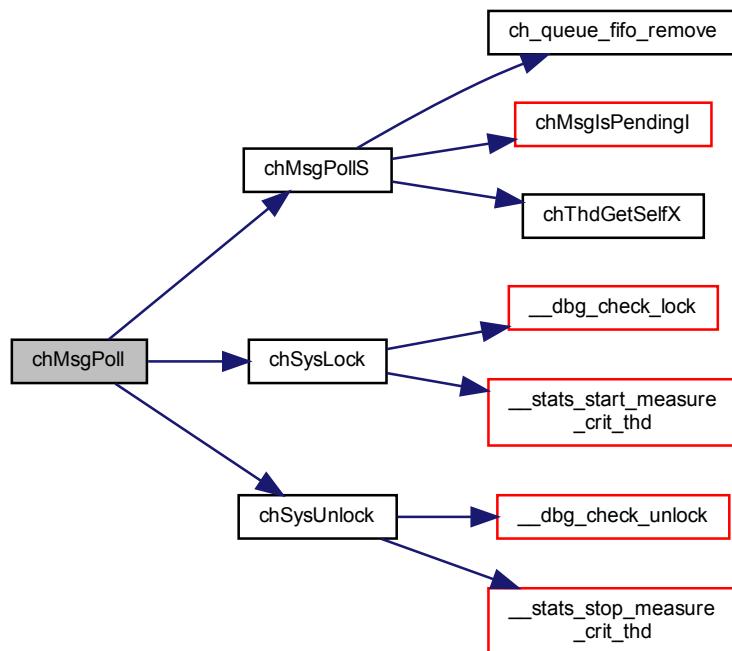
Return values

<code>NULL</code>	if no incoming message 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:



7.30.2.9 chMsgIsPendingI()

```
static bool chMsgIsPendingI (
    thread_t * tp ) [inline], [static]
```

Evaluates to `true` if the thread has pending messages.

Parameters

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

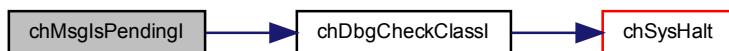
Returns

The pending messages status.

Function Class:

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

Here is the call graph for this function:



7.30.2.10 chMsgGet()

```
static msg_t chMsgGet (
    thread_t * tp ) [inline], [static]
```

Returns the message carried by the specified thread.

Precondition

This function must be invoked immediately after exiting a call to `chMsgWait()`.

Parameters

in	<code>tp</code>	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.

7.30.2.11 chMsgReleaseS()

```
static void chMsgReleaseS (
    thread_t * tp,
    msg_t msg ) [inline], [static]
```

Releases the thread waiting on top of the messages queue.

Precondition

Invoke this function only after a message has been received using [chMsgWait \(\)](#).

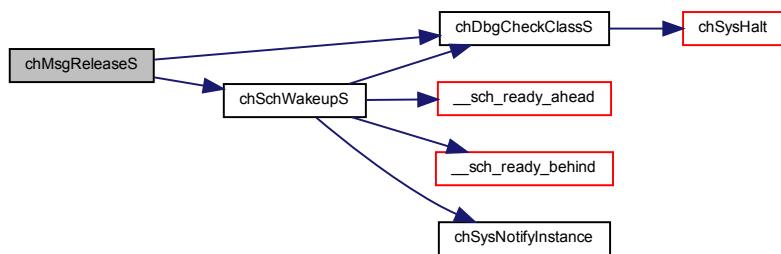
Parameters

in	<i>tp</i>	pointer to the thread
in	<i>msg</i>	message to be returned to the sender

Function Class:

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

Here is the call graph for this function:



7.31 Dynamic Threads

7.31.1 Detailed Description

Dynamic threads related APIs and services.

Functions

- `thread_t * chThdCreateFromHeap (memory_heap_t *heapp, size_t size, const char *name, tprio_t prio, tfunc_t pf, void *arg)`
Creates a new thread allocating the memory from the heap.
- `thread_t * chThdCreateFromMemoryPool (memory_pool_t *mp, const char *name, tprio_t prio, tfunc_t pf, void *arg)`
Creates a new thread allocating the memory from the specified memory pool.

7.31.2 Function Documentation

7.31.2.1 chThdCreateFromHeap()

```
thread_t * chThdCreateFromHeap (
    memory_heap_t * heapp,
    size_t size,
    const char * name,
    tprio_t prio,
    tfunc_t pf,
    void * arg )
```

Creates a new thread allocating the memory from the heap.

Precondition

The configuration options `CH_CFG_USE_DYNAMIC` and `CH_CFG_USE_HEAP` must be enabled in order to use this function.

Note

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

The memory allocated for the thread is not released automatically, it is responsibility of the creator thread to call `chThdWait ()` and then release the allocated memory.

Parameters

in	<code>heapp</code>	heap from which allocate the memory or <code>NULL</code> for the default heap
in	<code>size</code>	size of the working area to be allocated
in	<code>name</code>	thread name
in	<code>prio</code>	the priority level for the new thread
in	<code>pf</code>	the thread function
Chibios/RT	<code>arg</code>	an argument passed to the thread function. It can be <code>NULL</code> .

Returns

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

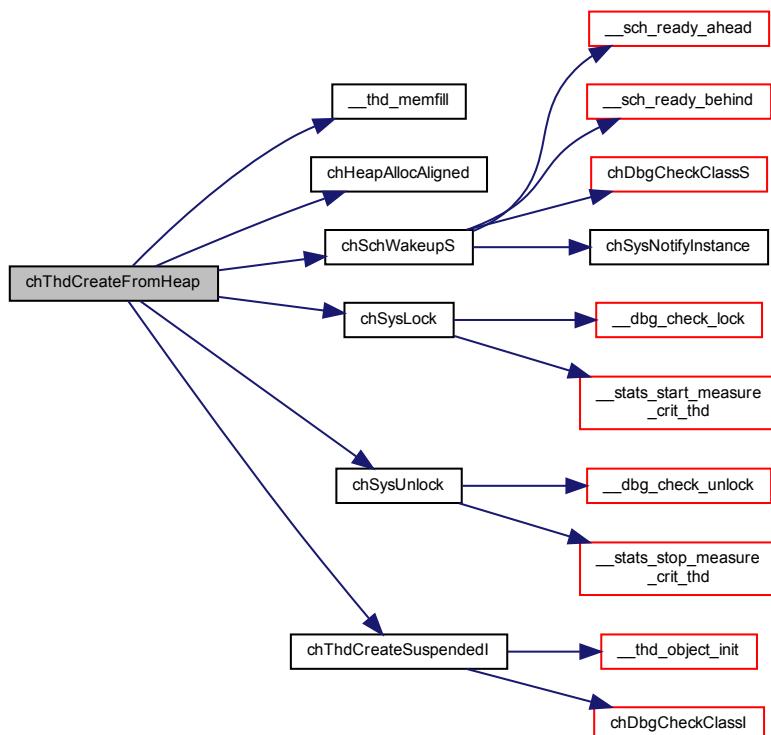
Return values

<code>NULL</code>	if the memory cannot be allocated.
-------------------	------------------------------------

Function Class:

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

Here is the call graph for this function:

**7.31.2.2 chThdCreateFromMemoryPool()**

```

thread_t * chThdCreateFromMemoryPool (
    memory_pool_t * mp,
    const char * name,
    tprio_t prio,
    tfunc_t pf,
    void * arg )
  
```

Creates a new thread allocating the memory from the specified memory pool.

Precondition

The configuration options `CH_CFG_USE_DYNAMIC` and `CH_CFG_USE_MEMPOOLS` must be enabled in order to use this function.

The pool must be initialized to contain only objects with alignment `PORT_WORKING_AREA_ALIGN`.

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 automatically, it is responsibility of the creator thread to call `chThdWait()` and then release the allocated memory.

Parameters

in	<i>mp</i>	pointer to the memory pool object
in	<i>name</i>	thread name
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 <code>NULL</code> .

Returns

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

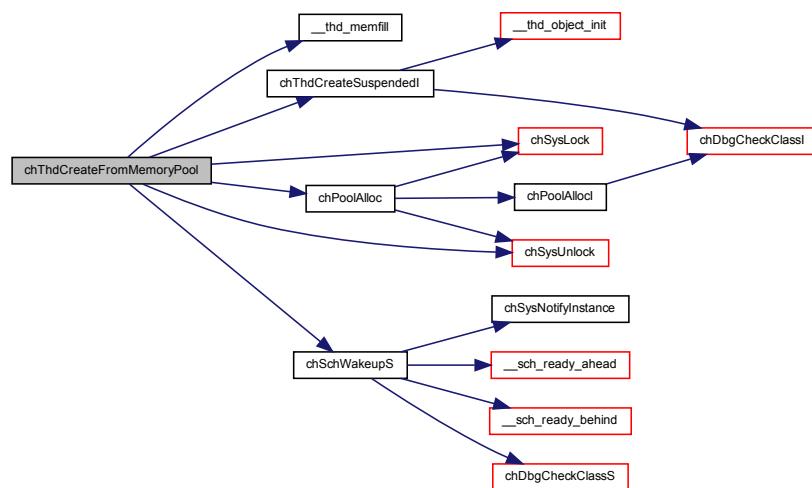
Return values

<code>NULL</code>	if the memory 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:



7.32 Registry

7.32.1 Detailed Description

Threads Registry related APIs and services.

Operation mode

The Threads Registry is a double linked list that holds all the active threads in the system.
Operations defined for the registry:

- **First**, returns the first, in creation order, active thread in the system.
- **Next**, returns the next, in creation order, active thread in the system.

The registry is meant to be mainly a debug feature, for example, using the registry a debugger can enumerate the active threads in any given moment or the shell can print the active threads and their state.
Another possible use is for centralized threads memory management, terminating threads can pulse an event source and an event handler can perform a scansion of the registry in order to recover the memory.

Precondition

In order to use the threads registry the `CH_CFG_USE_REGISTRY` option must be enabled in `chconf.h`.

Macros

- `#define REG_HEADER(oip) (&ch_system.reclist.queue)`
Access to the registry list header.
- `#define REG_REMOVE(tp) (void) ch_queue_dequeue(&(tp)->rqueue)`
Removes a thread from the registry list.
- `#define REG_INSERT(oip, tp) ch_queue_insert(REG_HEADER(oip), &(tp)->rqueue)`
Adds a thread to the registry list.

Data Structures

- struct `chdebug_t`

ChibiOS/RT memory signature record.

Functions

- `thread_t * chRegFirstThread (void)`
Returns the first thread in the system.
- `thread_t * chRegNextThread (thread_t *tp)`
Returns the thread next to the specified one.
- `thread_t * chRegFindThreadByName (const char *name)`
Retrieves a thread pointer by name.
- `thread_t * chRegFindThreadByPointer (thread_t *tp)`
Confirms that a pointer is a valid thread pointer.
- `thread_t * chRegFindThreadByWorkingArea (stkalign_t *wa)`
Confirms that a working area is being used by some active thread.
- `static void __reg_object_init (registry_t *rp)`
Initializes a registry.
- `static void chRegSetThreadName (const char *name)`
Sets the current thread name.
- `static const char * chRegGetThreadNameX (thread_t *tp)`
Returns the name of the specified thread.
- `static void chRegSetThreadNameX (thread_t *tp, const char *name)`
Changes the name of the specified thread.

7.32.2 Macro Definition Documentation

7.32.2.1 REG_HEADER

```
#define REG_HEADER(
    oip ) (&ch_system.reclist.queue)
```

Access to the registry list header.

7.32.2.2 REG_REMOVE

```
#define REG_REMOVE (
    tp ) (void) ch_queue_dequeue (&(tp)->rqueue)
```

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
----	----	------------------------------------

7.32.2.3 REG_INSERT

```
#define REG_INSERT(
    oip,
    tp ) ch_queue_insert(REG_HEADER(oip), &(tp)->rqueue)
```

Adds a thread to the registry list.

Note

This macro is not meant for use in application code.

Parameters

in	<i>oip</i>	pointer to the OS instance
in	<i>tp</i>	thread to add to the registry

7.32.3 Function Documentation

7.32.3.1 chRegFirstThread()

```
thread_t * chRegFirstThread (
    void )
```

Returns the first thread in the system.

Returns the most ancient thread in the system, usually this is the main thread unless it terminated. A reference is added to the returned thread in order to make sure its status is not lost.

Note

This function cannot return NULL because there is always at least one thread in the system.

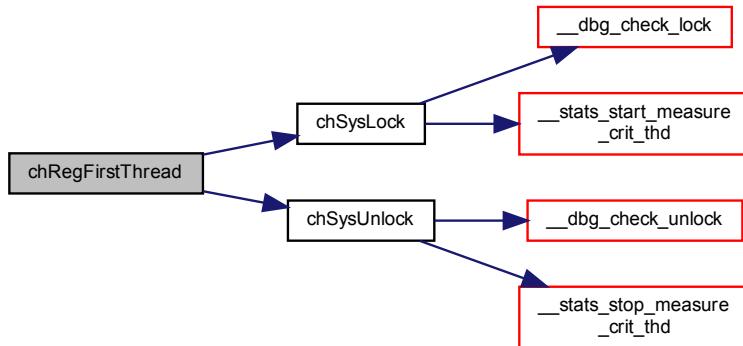
Returns

A reference to the most ancient thread.

Function Class:

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

Here is the call graph for this function:

**7.32.3.2 chRegNextThread()**

```
thread_t * chRegNextThread (
    thread_t * tp )
```

Returns the thread next to the specified one.

The reference counter of the specified thread is decremented and the reference counter of the returned thread is incremented.

Parameters

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

Returns

A reference to the next thread.

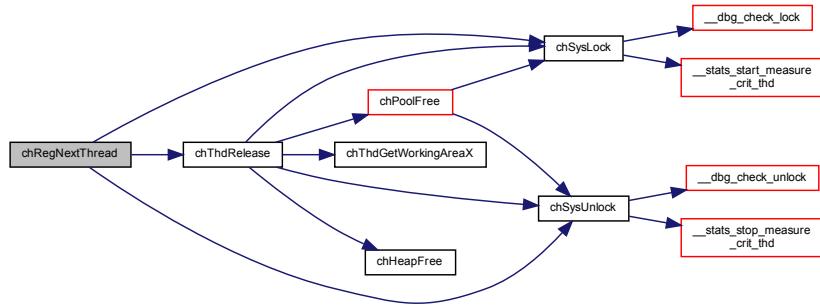
Return values

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:



7.32.3.3 `chRegFindThreadByName()`

```
thread_t * chRegFindThreadByName (
    const char * name )
```

Retrieves a thread pointer by name.

Note

The reference counter of the found thread is increased by one so it cannot be disposed incidentally after the pointer has been returned.

Parameters

in	<i>name</i>	the thread name
----	-------------	-----------------

Returns

A pointer to the found thread.

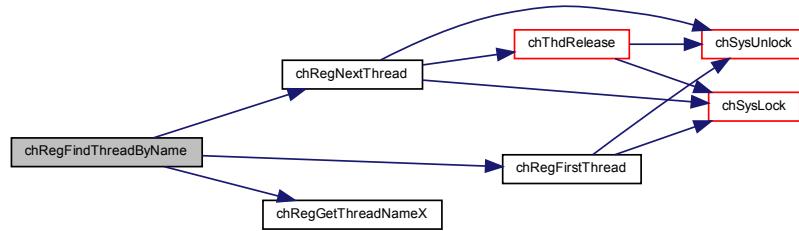
Return values

NULL	if a matching thread has not been found.
------	--

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:

**7.32.3.4 chRegFindThreadByPointer()**

```
thread_t * chRegFindThreadByPointer (
    thread_t * tp )
```

Confirms that a pointer is a valid thread pointer.

Note

The reference counter of the found thread is increased by one so it cannot be disposed incidentally after the pointer has been returned.

Parameters

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

Returns

A pointer to the found thread.

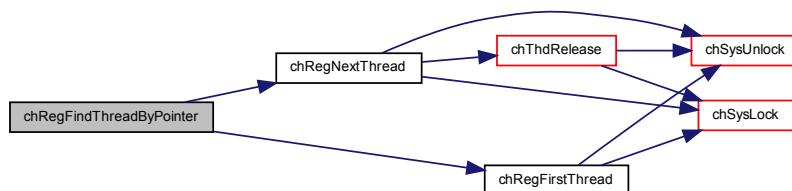
Return values

NULL	if a matching thread has not been found.
------	--

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:

**7.32.3.5 chRegFindThreadByWorkingArea()**

```
thread_t * chRegFindThreadByWorkingArea (
    stkalign_t * wa )
```

Confirms that a working area is being used by some active thread.

Note

The reference counter of the found thread is increased by one so it cannot be disposed incidentally after the pointer has been returned.

Parameters

in	<code>wa</code>	pointer to a static working area
----	-----------------	----------------------------------

Returns

A pointer to the found thread.

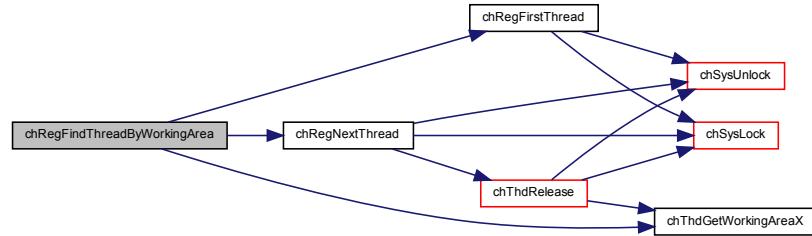
Return values

<code>NULL</code>	if a matching thread has not been found.
-------------------	--

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:

**7.32.3.6 __reg_object_init()**

```
static void __reg_object_init (
    registry_t * rp) [inline], [static]
```

Initializes a registry.

Note

Internal use only.

Parameters

out	<code>rp</code>	pointer to a <code>registry_t</code> structure
-----	-----------------	--

Function Class:

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

Here is the call graph for this function:



7.32.3.7 chRegSetThreadName()

```
static void chRegSetThreadName (
    const char * name ) [inline], [static]
```

Sets the current thread name.

Precondition

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

Parameters

in	<i>name</i>	thread name as a zero terminated string
----	-------------	---

Function Class:

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

7.32.3.8 chRegGetThreadNameX()

```
static const char* chRegGetThreadNameX (
    thread_t * tp ) [inline], [static]
```

Returns the name of the specified thread.

Precondition

This function only returns the pointer to the name if the option CH_CFG_USE_REGISTRY is enabled else NULL is returned.

Parameters

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

Returns

Thread name as a zero terminated string.

Return values

NULL	if the thread name has not been set.
------	--------------------------------------

7.32.3.9 chRegSetThreadNameX()

```
static void chRegSetThreadNameX (
    thread_t * tp,
    const char * name ) [inline], [static]
```

Changes the name of the specified thread.

Precondition

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

Parameters

in	<i>tp</i>	pointer to the thread
in	<i>name</i>	thread name as a zero terminated string

Function Class:

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

7.33 Debug

7.33.1 Detailed Description

Modules

- Checks and Assertions
- Tracing
- Statistics

7.34 Checks and Assertions

7.34.1 Detailed Description

Debug APIs and services:

- Runtime system state and call protocol check. The following panic messages can be generated:
 - SV#1, misplaced `chSysDisable()`.
 - * Called from an ISR.
 - * Called from a critical zone.
 - SV#2, misplaced `chSysSuspend()`
 - * Called from an ISR.
 - * Called from a critical zone.
 - SV#3, misplaced `chSysEnable()`.
 - * Called from an ISR.
 - * Called from a critical zone.
 - SV#4, misplaced `chSysLock()`.
 - * Called from an ISR.
 - * Called from a critical zone.
 - SV#5, misplaced `chSysUnlock()`.
 - * Called from an ISR.
 - * Not called from a critical zone.
 - SV#6, misplaced `chSysLockFromISR()`.
 - * Not called from an ISR.
 - * Called from a critical zone.
 - SV#7, misplaced `chSysUnlockFromISR()`.
 - * Not called from an ISR.
 - * Not called from a critical zone.
 - SV#8, misplaced `CH_IRQ_PROLOGUE()`.
 - * Not called at ISR begin.
 - * Called from a critical zone.
 - SV#9, misplaced `CH_IRQ_EPILOGUE()`.
 - * `CH_IRQ_PROLOGUE()` missing.
 - * Not called at ISR end.
 - * Called from a critical zone.
 - SV#10, misplaced I-class function.
 - * I-class function not called from within a critical zone.
 - SV#11, misplaced S-class function.
 - * S-class function not called from within a critical zone.
 - * Called from an ISR.
- Parameters check.
- Kernel assertions.

Note

Stack checks are not implemented in this module but in the port layer in an architecture-dependent way.

Debug related settings

- `#define CH_DBG_STACK_FILL_VALUE 0x55`
Fill value for thread stack area in debug mode.

Macro Functions

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

Typedefs

- `typedef struct ch_system_debug system_debug_t`
System debug data structure.

Data Structures

- `struct ch_system_debug`
System debug data structure.

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.
- `static void __dbg_object_init (system_debug_t *sdp)`
Debug support initialization.

7.34.2 Macro Definition Documentation

7.34.2.1 CH_DBG_STACK_FILL_VALUE

```
#define CH_DBG_STACK_FILL_VALUE 0x55
```

Fill value for thread stack area in debug mode.

7.34.2.2 chDbgCheck

```
#define chDbgCheck( c )
```

Value:

```
do { \
/*lint -save -e506 -e774 [2.1, 14.3] Can be a constant by design.*/
if (CH_DBG_ENABLE_CHECKS != FALSE) {
    if (unlikely(!(c))) {
/*lint -restore*/
        chSysHalt (__func__);
    }
} while (false)
```

Function parameters check.

If the condition check fails then the kernel panics and halts.

Note

The condition is tested only if the CH_DBG_ENABLE_CHECKS switch is specified in `chconf.h` else the macro does nothing.

Parameters

in	c	the condition to be verified to be true
----	---	---

Function Class:

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

7.34.2.3 chDbgAssert

```
#define chDbgAssert( c, r )
```

Value:

```

do {
    /*lint -save -e506 -e774 [2.1, 14.3] Can be a constant by design.*/
    if (CH_DBG_ENABLE_ASSERTS != FALSE) {
        if (unlikely(!(c))) {
            /*lint -restore*/
            chSysHalt(__func__);
        }
    }
} while (false)

```

Condition assertion.

If the condition check fails then the kernel panics with a message and halts.

Note

The condition is tested only if the CH_DBG_ENABLE_ASSERTS switch is specified in [chconf.h](#) else the macro does nothing.

The remark string is not currently used except for putting a comment in the code about the assertion.

Parameters

in	<i>c</i>	the condition to be verified to be true
in	<i>r</i>	a remark string

Function Class:

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

7.34.3 Typedef Documentation

7.34.3.1 `system_debug_t`

```
typedef struct ch_system_debug system_debug_t
```

System debug data structure.

7.34.4 Function Documentation

7.34.4.1 __dbg_check_disable()

```
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:



7.34.4.2 __dbg_check_suspend()

```
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:



7.34.4.3 __dbg_check_enable()

```
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:



7.34.4.4 __dbg_check_lock()

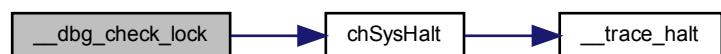
```
void __dbg_check_lock (
    void )
```

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

Function Class:

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

Here is the call graph for this function:



7.34.4.5 __dbg_check_unlock()

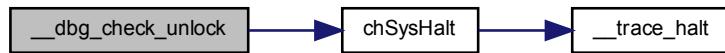
```
void __dbg_check_unlock (
    void )
```

Guard code for `chSysUnlock()`.

Function Class:

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

Here is the call graph for this function:



7.34.4.6 __dbg_check_lock_from_isr()

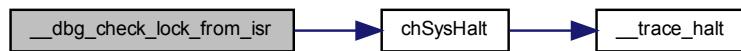
```
void __dbg_check_lock_from_isr (
    void )
```

Guard code for `chSysLockFromIsr()`.

Function Class:

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

Here is the call graph for this function:



7.34.4.7 __dbg_check_unlock_from_isr()

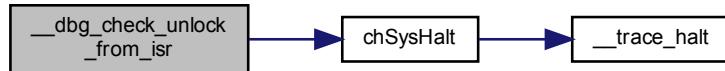
```
void __dbg_check_unlock_from_isr (
    void )
```

Guard code for `chSysUnlockFromIsr()`.

Function Class:

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

Here is the call graph for this function:



7.34.4.8 __dbg_check_enter_isr()

```
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:



7.34.4.9 __dbg_check_leave_isr()

```
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:



7.34.4.10 chDbgCheckClassI()

```
void chDbgCheckClassI (
    void )
```

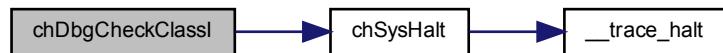
I-class functions context check.

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:



7.34.4.11 chDbgCheckClassS()

```
void chDbgCheckClassS (
    void )
```

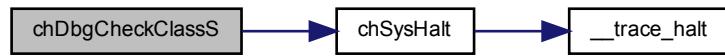
S-class functions context check.

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:



7.34.4.12 __dbg_object_init()

```
static void __dbg_object_init (
    system_debug_t * sdp ) [inline], [static]
```

Debug support initialization.

Note

Internal use only.

Parameters

out	sdp	pointer to the system_debug_t structure
-----	-----	---

Function Class:

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

7.35 Tracing

7.35.1 Detailed Description

System events tracing service.

Trace record types

- #define **CH_TRACE_TYPE_UNUSED** 0U
- #define **CH_TRACE_TYPE_READY** 1U
- #define **CH_TRACE_TYPE_SWITCH** 2U
- #define **CH_TRACE_TYPE_ISR_ENTER** 3U
- #define **CH_TRACE_TYPE_ISR_LEAVE** 4U
- #define **CH_TRACE_TYPE_HALT** 5U
- #define **CH_TRACE_TYPE_USER** 6U

Events to trace

- #define **CH_DBG_TRACE_MASK_DISABLED** 255U
- #define **CH_DBG_TRACE_MASK_NONE** 0U
- #define **CH_DBG_TRACE_MASK_READY** 1U
- #define **CH_DBG_TRACE_MASK_SWITCH** 2U
- #define **CH_DBG_TRACE_MASK_ISR** 4U
- #define **CH_DBG_TRACE_MASK_HALT** 8U
- #define **CH_DBG_TRACE_MASK_USER** 16U
- #define **CH_DBG_TRACE_MASK_SLOW**
- #define **CH_DBG_TRACE_MASK_ALL**

Debug related settings

- #define **CH_DBG_TRACE_MASK** CH_DBG_TRACE_MASK_DISABLED
Trace buffer entries.
- #define **CH_DBG_TRACE_BUFFER_SIZE** 128
Trace buffer entries.

Data Structures

- struct **trace_event_t**
Trace buffer record.
- struct **trace_buffer_t**
Trace buffer header.

Functions

- static NOINLINE void `trace_next (os_instance_t *oip)`
Writes a time stamp and increases the trace buffer pointer.
- void `__trace_object_init (trace_buffer_t *tbp)`
Circular trace buffer initialization.
- void `__trace_ready (thread_t *tp, msg_t msg)`
Inserts in the circular debug trace buffer a ready record.
- void `__trace_switch (thread_t *ntp, thread_t *otp)`
Inserts in the circular debug trace buffer a context switch record.
- void `__trace_isr_enter (const char *isr)`
Inserts in the circular debug trace buffer an ISR-enter record.
- void `__trace_isr_leave (const char *isr)`
Inserts in the circular debug trace buffer an ISR-leave record.
- void `__trace_halt (const char *reason)`
Inserts in the circular debug trace buffer an halt record.
- void `chTraceWritel (void *up1, void *up2)`
Adds an user trace record to the trace buffer.
- void `chTraceWrite (void *up1, void *up2)`
Adds an user trace record to the trace buffer.
- void `chTraceSuspendl (uint16_t mask)`
Suspends one or more trace events.
- void `chTraceSuspend (uint16_t mask)`
Suspends one or more trace events.
- void `chTraceResumel (uint16_t mask)`
Resumes one or more trace events.
- void `chTraceResume (uint16_t mask)`
Resumes one or more trace events.

7.35.2 Macro Definition Documentation

7.35.2.1 CH_DBG_TRACE_MASK

```
#define CH_DBG_TRACE_MASK CH_DBG_TRACE_MASK_DISABLED
```

Trace buffer entries.

7.35.2.2 CH_DBG_TRACE_BUFFER_SIZE

```
#define CH_DBG_TRACE_BUFFER_SIZE 128
```

Trace buffer entries.

Note

The trace buffer is only allocated if CH_DBG_TRACE_MASK is different from CH_DBG_TRACE_MASK_DISABLED.

7.35.3 Function Documentation

7.35.3.1 trace_next()

```
static NOINLINE void trace_next (
    os_instance_t * oip ) [static]
```

Writes a time stamp and increases the trace buffer pointer.

Function Class:

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

Here is the call graph for this function:



7.35.3.2 __trace_object_init()

```
void __trace_object_init (
    trace_buffer_t * tbp )
```

Circular trace buffer initialization.

Note

Internal use only.

Parameters

out	tbp	pointer to the <code>trace_buffer_t</code> structure
-----	-----	--

Function Class:

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

7.35.3.3 __trace_ready()

```
void __trace_ready (
    thread_t * tp,
    msg_t msg )
```

Inserts in the circular debug trace buffer a ready record.

Parameters

in	<i>tp</i>	the thread that just became ready
in	<i>msg</i>	the thread ready message

Function Class:

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

7.35.3.4 __trace_switch()

```
void __trace_switch (
    thread_t * ntp,
    thread_t * otp )
```

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

Parameters

in	<i>ntp</i>	the thread being switched in
in	<i>otp</i>	the thread being switched out

Function Class:

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

7.35.3.5 __trace_isr_enter()

```
void __trace_isr_enter (
    const char * isr )
```

Inserts in the circular debug trace buffer an ISR-enter record.

Parameters

in	<i>isr</i>	name of the isr
----	------------	-----------------

Function Class:

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

7.35.3.6 __trace_isr_leave()

```
void __trace_isr_leave (
    const char * isr )
```

Inserts in the circular debug trace buffer an ISR-leave record.

Parameters

in	<i>isr</i>	name of the isr
----	------------	-----------------

Function Class:

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

7.35.3.7 __trace_halt()

```
void __trace_halt (
    const char * reason )
```

Inserts in the circular debug trace buffer an halt record.

Parameters

in	<i>reason</i>	the halt error string
----	---------------	-----------------------

Function Class:

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

7.35.3.8 chTraceWriteI()

```
void chTraceWriteI (
    void * up1,
    void * up2 )
```

Adds an user trace record to the trace buffer.

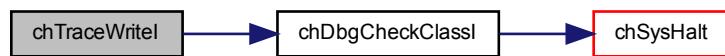
Parameters

in	<i>up1</i>	user parameter 1
in	<i>up2</i>	user parameter 2

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:

**7.35.3.9 chTraceWrite()**

```
void chTraceWrite (
    void * up1,
    void * up2 )
```

Adds an user trace record to the trace buffer.

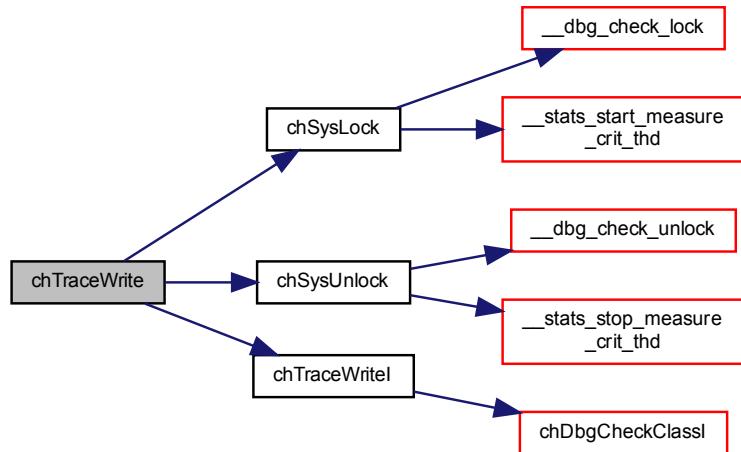
Parameters

in	<i>up1</i>	user parameter 1
in	<i>up2</i>	user parameter 2

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:



7.35.3.10 chTraceSuspendI()

```
void chTraceSuspendI (
    uint16_t mask )
```

Suspends one or more trace events.

Parameters

in	mask	mask of the trace events to be suspended
----	------	--

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:



7.35.3.11 chTraceSuspend()

```
void chTraceSuspend (
    uint16_t mask )
```

Suspends one or more trace events.

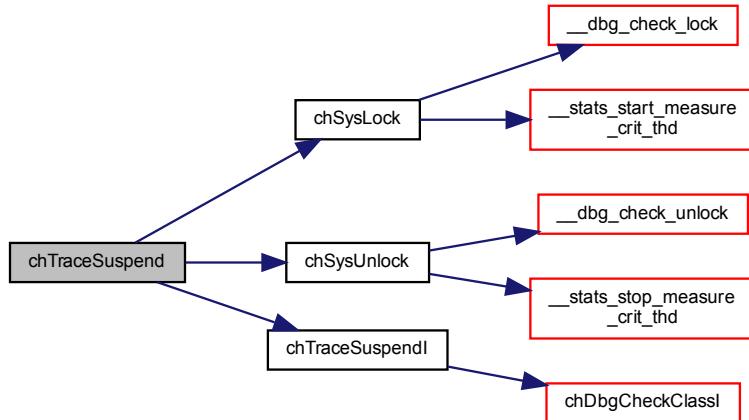
Parameters

in	mask	mask of the trace events to be suspended
----	------	--

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:



7.35.3.12 chTraceResumeI()

```
void chTraceResumeI (
    uint16_t mask )
```

Resumes one or more trace events.

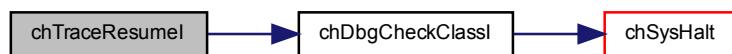
Parameters

in	<i>mask</i>	mask of the trace events to be resumed
----	-------------	--

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:



7.35.3.13 chTraceResume()

```
void chTraceResume (
    uint16_t mask )
```

Resumes one or more trace events.

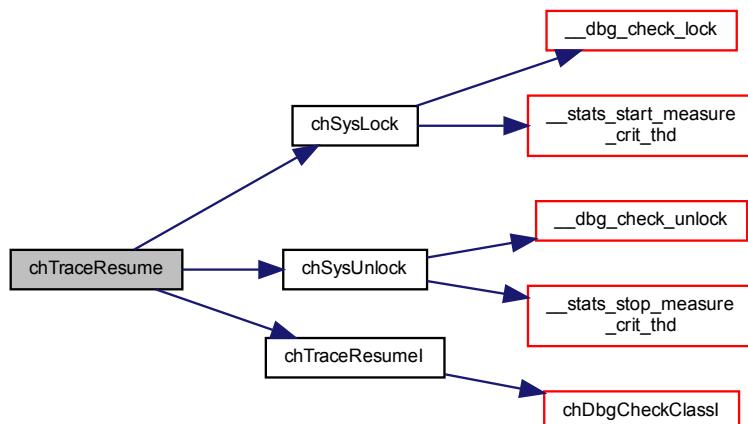
Parameters

in	mask	mask of the trace events to be resumed
----	------	--

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:



7.36 Statistics

7.36.1 Detailed Description

Statistics services.

Data Structures

- struct `kernel_stats_t`
Type of a kernel statistics structure.

Functions

- void `__stats_increase_irq(void)`
Increases the IRQ counter.
- void `__stats_ctxswc(thread_t *ntp, thread_t *otp)`
Updates context switch related statistics.
- void `__stats_start_measure_crit_thd(void)`
Starts the measurement of a thread critical zone.
- void `__stats_stop_measure_crit_thd(void)`
Stops the measurement of a thread critical zone.
- void `__stats_start_measure_crit_isr(void)`
Starts the measurement of an ISR critical zone.
- void `__stats_stop_measure_crit_isr(void)`
Stops the measurement of an ISR critical zone.
- static void `__stats_object_init(kernel_stats_t *ksp)`
Statistics initialization.

7.36.2 Function Documentation

7.36.2.1 `__stats_increase_irq()`

```
void __stats_increase_irq (
    void )
```

Increases the IRQ counter.

7.36.2.2 `__stats_ctxswc()`

```
void __stats_ctxswc (
    thread_t * ntp,
    thread_t * otp )
```

Updates context switch related statistics.

Parameters

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

Here is the call graph for this function:

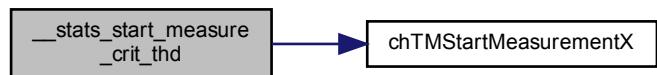


7.36.2.3 __stats_start_measure_crit_thd()

```
void __stats_start_measure_crit_thd (
    void )
```

Starts the measurement of a thread critical zone.

Here is the call graph for this function:

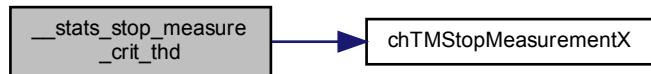


7.36.2.4 __stats_stop_measure_crit_thd()

```
void __stats_stop_measure_crit_thd (
    void )
```

Stops the measurement of a thread critical zone.

Here is the call graph for this function:

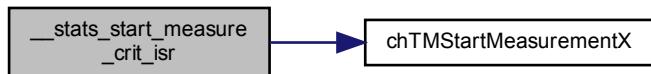


7.36.2.5 `__stats_start_measure_crit_isr()`

```
void __stats_start_measure_crit_isr (\n    void )
```

Starts the measurement of an ISR critical zone.

Here is the call graph for this function:



7.36.2.6 `__stats_stop_measure_crit_isr()`

```
void __stats_stop_measure_crit_isr (\n    void )
```

Stops the measurement of an ISR critical zone.

Here is the call graph for this function:



7.36.2.7 __stats_object_init()

```
static void __stats_object_init (
    kernel_stats_t * ksp ) [inline], [static]
```

Statistics initialization.

Note

Internal use only.

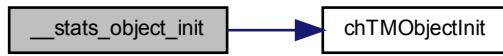
Parameters

out	<i>ksp</i>	pointer to the kernel__stats_t structure
-----	------------	--

Function Class:

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

Here is the call graph for this function:



7.37 OS Library

7.37.1 Detailed Description

The OS Library is a set of RTOS extensions compatible with both the RT and NIL RTOSes.

Modules

- [Version Numbers and Identification](#)
- [Synchronization](#)
- [Memory Management](#)
- [Complex Services](#)

7.38 Version Numbers and Identification

7.38.1 Detailed Description

OS Library related info.

Macros

- `#define __CHIBIOS_OSLIB__`
ChibiOS/LIB identification macro.
- `#define CH_OSLIB_STABLE 0`
Stable release flag.

ChibiOS/LIB version identification

- `#define CH_OSLIB_VERSION "1.3.0"`
OS Library version string.
- `#define CH_OSLIB_MAJOR 1`
OS Library version major number.
- `#define CH_OSLIB_MINOR 3`
OS Library version minor number.
- `#define CH_OSLIB_PATCH 0`
OS Library version patch number.

Functions

- `static void __oslib_init (void)`
Initialization of all library modules.

7.38.2 Macro Definition Documentation

7.38.2.1 __CHIBIOS_OSLIB__

```
#define __CHIBIOS_OSLIB__
```

ChibiOS/LIB identification macro.

7.38.2.2 CH_OSLIB_STABLE

```
#define CH_OSLIB_STABLE 0
```

Stable release flag.

7.38.2.3 CH_OSLIB_VERSION

```
#define CH_OSLIB_VERSION "1.3.0"
```

OS Library version string.

7.38.2.4 CH_OSLIB_MAJOR

```
#define CH_OSLIB_MAJOR 1
```

OS Library version major number.

7.38.2.5 CH_OSLIB_MINOR

```
#define CH_OSLIB_MINOR 3
```

OS Library version minor number.

7.38.2.6 CH_OSLIB_PATCH

```
#define CH_OSLIB_PATCH 0
```

OS Library version patch number.

7.38.3 Function Documentation

7.38.3.1 __oslib_init()

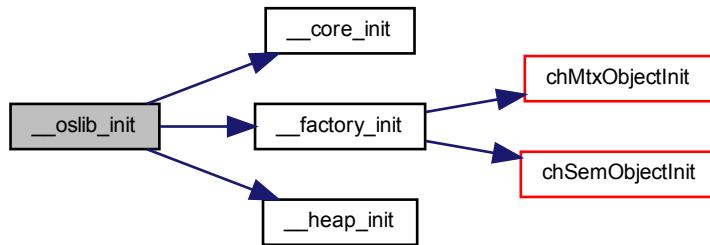
```
static void __oslib_init (
    void ) [inline], [static]
```

Initialization of all library modules.

Function Class:

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

Here is the call graph for this function:



7.39 Synchronization

7.39.1 Detailed Description

Synchronization services.

Modules

- [Binary Semaphores](#)
- [Mailboxes](#)
- [Pipes](#)
- [Delegate Threads](#)
- [Jobs Queues](#)

7.40 Binary Semaphores

7.40.1 Detailed Description

Macros

- `#define __BSEMAPHORE_DATA(name, taken) {__SEMAPHORE_DATA(name.sem, ((taken) ? 0 : 1))}`
Data part of a static semaphore initializer.
- `#define BSEMAPHORE_DECL(name, taken) binary_semaphore_t name = __BSEMAPHORE_DATA(name, taken)`
Static semaphore initializer.

Typedefs

- `typedef struct ch_binary_semaphore binary_semaphore_t`
Binary semaphore type.

Data Structures

- `struct ch_binary_semaphore`
Binary semaphore type.

Functions

- `static void chBSemObjectInit (binary_semaphore_t *bsp, bool taken)`
Initializes a binary semaphore.
- `static msg_t chBSemWait (binary_semaphore_t *bsp)`
Wait operation on the binary semaphore.
- `static msg_t chBSemWaitS (binary_semaphore_t *bsp)`
Wait operation on the binary semaphore.
- `static msg_t chBSemWaitTimeout (binary_semaphore_t *bsp, sysinterval_t timeout)`
Wait operation on the binary semaphore.
- `static msg_t chBSemWaitTimeout (binary_semaphore_t *bsp, sysinterval_t timeout)`
Wait operation on the binary semaphore.
- `static void chBSemResetl (binary_semaphore_t *bsp, bool taken)`
Reset operation on the binary semaphore.
- `static void chBSemReset (binary_semaphore_t *bsp, bool taken)`
Reset operation on the binary semaphore.
- `static void chBSemSignall (binary_semaphore_t *bsp)`
Performs a signal operation on a binary semaphore.
- `static void chBSemSignal (binary_semaphore_t *bsp)`
Performs a signal operation on a binary semaphore.
- `static bool chBSemGetStatel (const binary_semaphore_t *bsp)`
Returns the binary semaphore current state.

7.40.2 Macro Definition Documentation

7.40.2.1 __BSEMAPHORE_DATA

```
#define __BSEMAPHORE_DATA(
    name,
    taken ) {__SEMAPHORE_DATA(name.sem, ((taken) ? 0 : 1))}
```

Data part of a static semaphore initializer.

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

Parameters

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

7.40.2.2 BSEMAPHORE_DECL

```
#define BSEMAPHORE_DECL (
    name,
    taken ) binary_semaphore_t name = __BSEMAPHORE_DATA(name, taken)
```

Static semaphore initializer.

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

Parameters

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

7.40.3 Typedef Documentation

7.40.3.1 binary_semaphore_t

```
typedef struct ch_binary_semaphore binary_semaphore_t
```

Binary semaphore type.

7.40.4 Function Documentation

7.40.4.1 chBSemObjectInit()

```
static void chBSemObjectInit (
    binary_semaphore_t * bsp,
    bool taken ) [inline], [static]
```

Initializes a binary semaphore.

Parameters

out	<i>bsp</i>	pointer to a <code>binary_semaphore_t</code> structure
in	<i>taken</i>	initial state of the binary semaphore: <ul style="list-style-type: none"> • <i>false</i>, the initial state is not taken. • <i>true</i>, the initial state is taken.

Function Class:

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

Here is the call graph for this function:



7.40.4.2 chBSemWait()

```
static msg_t chBSemWait (
    binary_semaphore_t * bsp ) [inline], [static]
```

Wait operation on the binary semaphore.

Parameters

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

Returns

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

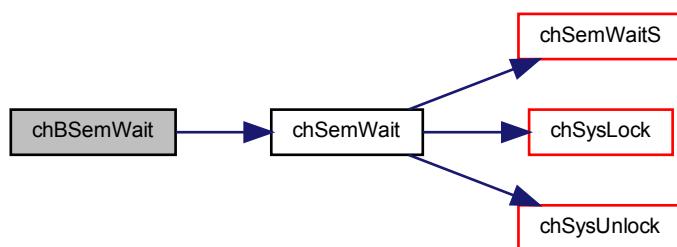
Return values

<i>MSG_OK</i>	if the binary semaphore has been successfully taken.
<i>MSG_RESET</i>	if the binary semaphore has been reset using <code>bsemReset()</code> .

Function Class:

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

Here is the call graph for this function:

7.40.4.3 `chBSemWaitS()`

```
static msg_t chBSemWaitS (
    binary_semaphore_t * bsp ) [inline], [static]
```

Wait operation on the binary semaphore.

Parameters

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

Returns

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

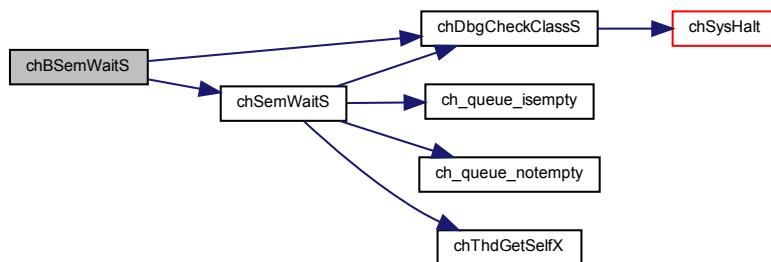
Return values

<i>MSG_OK</i>	if the binary semaphore has been successfully taken.
<i>MSG_RESET</i>	if the binary semaphore has been reset using <code>bsemReset()</code> .

Function Class:

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

Here is the call graph for this function:



7.40.4.4 chBSemWaitTimeoutS()

```
static msg_t chBSemWaitTimeouts (
    binary_semaphore_t * bsp,
    sysinterval_t timeout ) [inline], [static]
```

Wait operation on the binary semaphore.

Parameters

in	<i>bsp</i>	pointer to a <code>binary_semaphore_t</code> structure
in	<i>timeout</i>	the number of ticks before the operation timeouts, the following special values are allowed: <ul style="list-style-type: none"> • <code>TIME_IMMEDIATE</code> immediate timeout. • <code>TIME_INFINITE</code> no timeout.

Returns

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

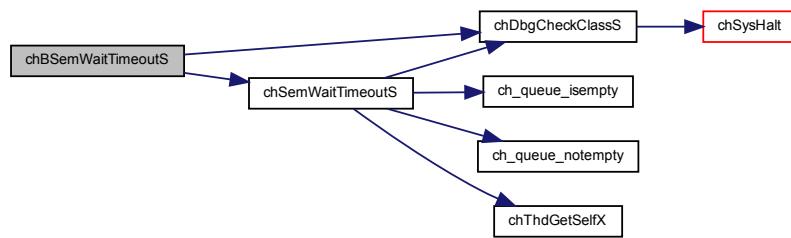
Return values

<code>MSG_OK</code>	if the binary semaphore has been successfully taken.
<code>MSG_RESET</code>	if the binary semaphore has been reset using <code>bsemReset ()</code> .
<code>MSG_TIMEOUT</code>	if the binary semaphore has not been signaled or reset within the specified timeout.

Function Class:

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

Here is the call graph for this function:



7.40.4.5 chBSemWaitTimeout()

```
static msg_t chBSemWaitTimeout (
    binary_semaphore_t * bsp,
    sysinterval_t timeout ) [inline], [static]
```

Wait operation on the binary semaphore.

Parameters

in	<i>bsp</i>	pointer to a <code>binary_semaphore_t</code> structure
in	<i>timeout</i>	the number of ticks before the operation timeouts, the following special values are allowed: <ul style="list-style-type: none"> • <code>TIME_IMMEDIATE</code> immediate timeout. • <code>TIME_INFINITE</code> no timeout.

Returns

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

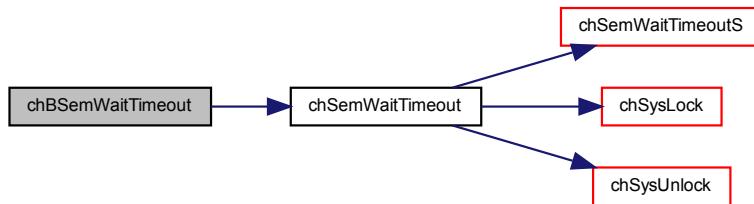
Return values

<code>MSG_OK</code>	if the binary semaphore has been successfully taken.
<code>MSG_RESET</code>	if the binary semaphore has been reset using <code>bsemReset()</code> .
<code>MSG_TIMEOUT</code>	if the binary semaphore has not been signaled or reset within the specified timeout.

Function Class:

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

Here is the call graph for this function:

**7.40.4.6 chBSemResetI()**

```
static void chBSemResetI (
    binary_semaphore_t * bsp,
    bool taken ) [inline], [static]
```

Reset operation on the binary semaphore.

Note

The released threads can recognize they were waked up by a reset rather than a signal because the `bsem->Wait()` will return `MSG_RESET` instead of `MSG_OK`.

This function does not reschedule.

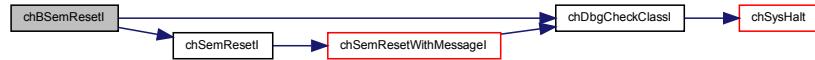
Parameters

in	<i>bsp</i>	pointer to a <code>binary_semaphore_t</code> structure
in	<i>taken</i>	new state of the binary semaphore <ul style="list-style-type: none"> • <i>false</i>, the new state is not taken. • <i>true</i>, the new state is taken.

Function Class:

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:

**7.40.4.7 chBSemReset()**

```

static void chBSemReset (
    binary_semaphore_t * bsp,
    bool taken ) [inline], [static]
  
```

Reset operation on the binary semaphore.

Note

The released threads can recognize they were wakened up by a reset rather than a signal because the `bsem->Wait()` will return `MSG_RESET` instead of `MSG_OK`.

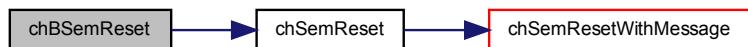
Parameters

in	<i>bsp</i>	pointer to a <code>binary_semaphore_t</code> structure
in	<i>taken</i>	new state of the binary semaphore <ul style="list-style-type: none"> • <i>false</i>, the new state is not taken. • <i>true</i>, the new state is taken.

Function Class:

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

Here is the call graph for this function:



7.40.4.8 chBSemSignall()

```
static void chBSemSignall (
    binary_semaphore_t * bsp ) [inline], [static]
```

Performs a signal operation on a binary semaphore.

Note

This function does not reschedule.

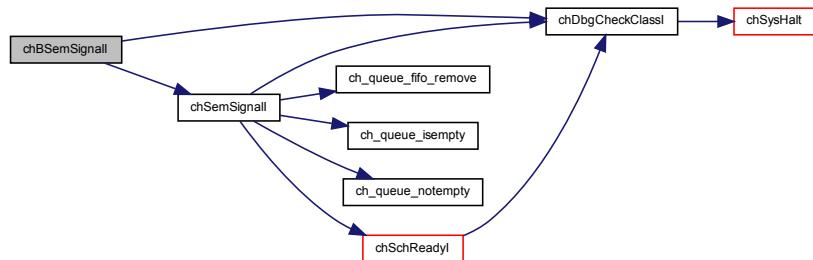
Parameters

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

Function Class:

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

Here is the call graph for this function:



7.40.4.9 chBSemSignal()

```
static void chBSemSignal (
    binary_semaphore_t * bsp ) [inline], [static]
```

Performs a signal operation on a binary semaphore.

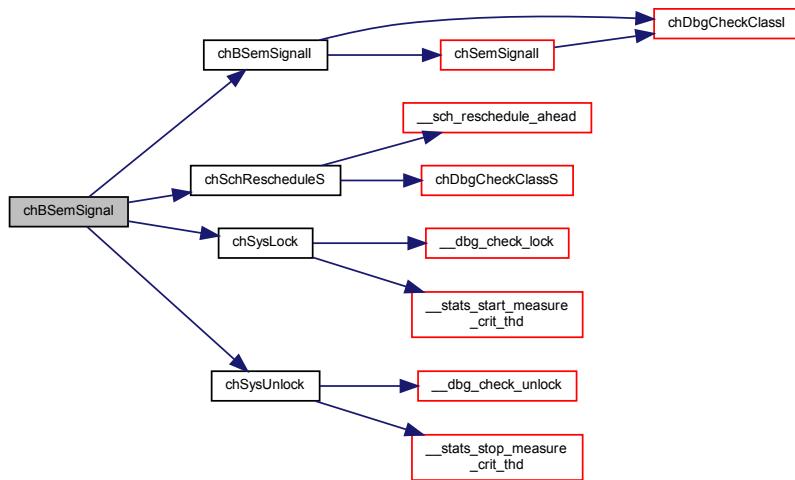
Parameters

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

Function Class:

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

Here is the call graph for this function:

**7.40.4.10 chBSemGetStateI()**

```
static bool chBSemGetStateI (
    const binary_semaphore_t * bsp ) [inline], [static]
```

Returns the binary semaphore current state.

Parameters

in	<code>bsp</code>	pointer to a <code>binary_semaphore_t</code> structure
----	------------------	--

Returns

The binary semaphore current state.

Return values

<code>false</code>	if the binary semaphore is not taken.
<code>true</code>	if the binary semaphore is taken.

Function Class:

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

Here is the call graph for this function:



7.41 Mailboxes

7.41.1 Detailed Description

Asynchronous messages.

Operation mode

A mailbox is an asynchronous communication mechanism.

Operations defined for mailboxes:

- **Post:** Posts a message on the mailbox in FIFO order.
- **Post Ahead:** Posts a message on the mailbox with urgent priority.
- **Fetch:** A message is fetched from the mailbox and removed from the queue.
- **Reset:** The mailbox is emptied and all the stored messages are lost.

A message is a variable of type `msg_t` that is guaranteed to have the same size of and be compatible with (data) pointers (anyway an explicit cast is needed). If larger messages need to be exchanged then a pointer to a structure can be posted in the mailbox but the posting side has no predefined way to know when the message has been processed. A possible approach is to allocate memory (from a memory pool for example) from the posting side and free it on the fetching side. Another approach is to set a "done" flag into the structure pointed by the message.

Precondition

In order to use the mailboxes APIs the `CH_CFG_USE_MAILBOXES` option must be enabled in `chconf.h`.

Note

Compatible with RT and NIL.

Macros

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

Data Structures

- `struct mailbox_t`
Structure representing a mailbox object.

Functions

- void `chMBOBJECTInit (mailbox_t *mbp, msg_t *buf, size_t n)`
Initializes a `mailbox_t` object.
- void `chMBReset (mailbox_t *mbp)`
Resets a `mailbox_t` object.
- void `chMBResetl (mailbox_t *mbp)`
Resets a `mailbox_t` object.
- msg_t `chMBPostTimeout (mailbox_t *mbp, msg_t msg, sysinterval_t timeout)`
Posts a message into a mailbox.
- msg_t `chMBPostTimeoutS (mailbox_t *mbp, msg_t msg, sysinterval_t timeout)`
Posts a message into a mailbox.
- msg_t `chMBPostl (mailbox_t *mbp, msg_t msg)`
Posts a message into a mailbox.
- msg_t `chMBPostAheadTimeout (mailbox_t *mbp, msg_t msg, sysinterval_t timeout)`
Posts an high priority message into a mailbox.
- msg_t `chMBPostAheadTimeoutS (mailbox_t *mbp, msg_t msg, sysinterval_t timeout)`
Posts an high priority message into a mailbox.
- msg_t `chMBPostAheadl (mailbox_t *mbp, msg_t msg)`
Posts an high priority message into a mailbox.
- msg_t `chMBFetchTimeout (mailbox_t *mbp, msg_t *msgp, sysinterval_t timeout)`
Retrieves a message from a mailbox.
- msg_t `chMBFetchTimeoutS (mailbox_t *mbp, msg_t *msgp, sysinterval_t timeout)`
Retrieves a message from a mailbox.
- msg_t `chMBFetchl (mailbox_t *mbp, msg_t *msgp)`
Retrieves a message from a mailbox.
- static size_t `chMBGetSizel (const mailbox_t *mbp)`
Returns the mailbox buffer size as number of messages.
- static size_t `chMBGetUsedCountl (const mailbox_t *mbp)`
Returns the number of used message slots into a mailbox.
- static size_t `chMBGetFreeCountl (const mailbox_t *mbp)`
Returns the number of free message slots into a mailbox.
- static msg_t `chMBPeekl (const mailbox_t *mbp)`
Returns the next message in the queue without removing it.
- static void `chMBResumeX (mailbox_t *mbp)`
Terminates the reset state.

7.41.2 Macro Definition Documentation

7.41.2.1 __MAILBOX_DATA

```
#define __MAILBOX_DATA(
    name,
    buffer,
    size )
```

Value:

```
{
    \
    (msg_t *) (buffer),
    (msg_t *) (buffer) + size,
    (msg_t *) (buffer),
    (msg_t *) (buffer),
    (size_t) 0,
    false,
    __THREADS_QUEUE_DATA(name.qw),
    __THREADS_QUEUE_DATA(name.qr),
}
```

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 array of msg_t
in	<i>size</i>	number of msg_t elements in the buffer array

7.41.2.2 MAILBOX_DECL

```
#define MAILBOX_DECL(
    name,
    buffer,
    size ) mailbox_t name = __MAILBOX_DATA(name, buffer, size)
```

Static mailbox initializer.

Statically initialized mailboxes require no explicit initialization using [chMBOBJECTINIT\(\)](#).

Parameters

in	<i>name</i>	the name of the mailbox variable
in	<i>buffer</i>	pointer to the mailbox buffer array of msg_t
in	<i>size</i>	number of msg_t elements in the buffer array

7.41.3 Function Documentation

7.41.3.1 chMBOBJECTINIT()

```
void chMBOBJECTINIT (
    mailbox_t * mbp,
    msg_t * buf,
    size_t n )
```

Initializes a `mailbox_t` object.

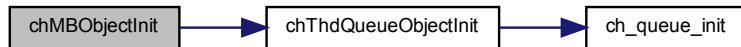
Parameters

out	<code>mbp</code>	the pointer to the <code>mailbox_t</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:



7.41.3.2 chMBRESET()

```
void chMBRESET (
    mailbox_t * mbp )
```

Resets a `mailbox_t` object.

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

Postcondition

The mailbox is in reset state, all operations will fail and return `MSG_RESET` until the mailbox is enabled again using `chMBRESUME()`.

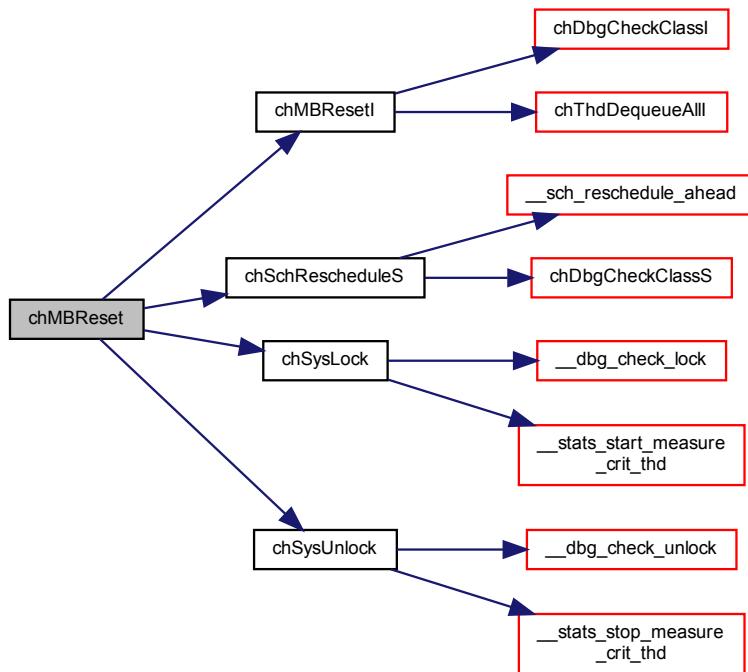
Parameters

in	<code>mbp</code>	the pointer to an initialized <code>mailbox_t</code> object
----	------------------	---

Function Class:

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

Here is the call graph for this function:



7.41.3.3 chMBResetI()

```
void chMBResetI (
    mailbox_t * mbp )
```

Resets a `mailbox_t` object.

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

Postcondition

The mailbox is in reset state, all operations will fail and return `MSG_RESET` until the mailbox is enabled again using `chMBResumeX()`.

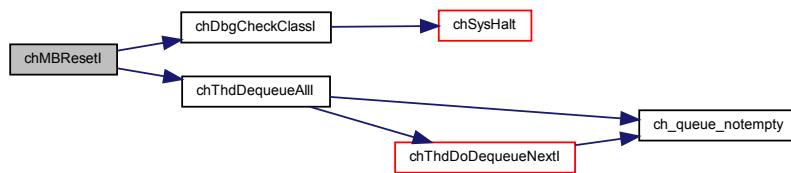
Parameters

in	<code>mbp</code>	the pointer to an initialized <code>mailbox_t</code> object
----	------------------	---

Function Class:

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

Here is the call graph for this function:

**7.41.3.4 chMBPostTimeout()**

```
msg_t chMBPostTimeout (
    mailbox_t * mbp,
    msg_t msg,
    sysinterval_t timeout )
```

Posts a message into a mailbox.

The invoking thread waits until a empty slot in the mailbox becomes available or the specified time runs out.

Parameters

in	<i>mbp</i>	the pointer to an initialized <code>mailbox_t</code> object
in	<i>msg</i>	the message to be posted on the mailbox
in	<i>timeout</i>	the number of ticks before the operation timeouts, the following special values are allowed: <ul style="list-style-type: none"> • <code>TIME_IMMEDIATE</code> immediate timeout. • <code>TIME_INFINITE</code> no timeout.

Returns

The operation status.

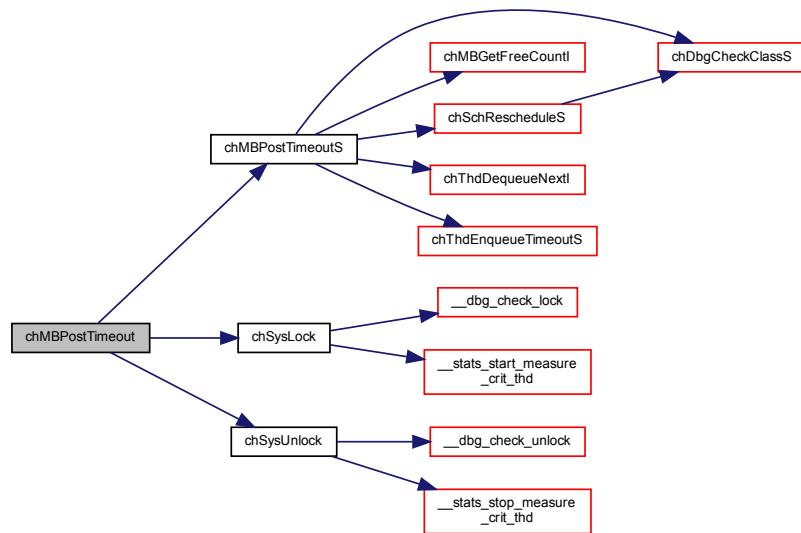
Return values

<code>MSG_OK</code>	if a message has been correctly posted.
<code>MSG_RESET</code>	if the mailbox has been reset.
<code>MSG_TIMEOUT</code>	if the operation has timed out.

Function Class:

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

Here is the call graph for this function:



7.41.3.5 chMBPostTimeoutS()

```

msg_t chMBPostTimeoutS (
    mailbox_t * mbp,
    msg_t msg,
    sysinterval_t timeout )
  
```

Posts a message into a mailbox.

The invoking thread waits until a empty slot in the mailbox becomes available or the specified time runs out.

Parameters

in	<code>mbp</code>	the pointer to an initialized <code>mailbox_t</code> object
in	<code>msg</code>	the message to be posted on the mailbox
in	<code>timeout</code>	the number of ticks before the operation timeouts, the following special values are allowed: <ul style="list-style-type: none"> • <code>TIME_IMMEDIATE</code> immediate timeout. • <code>TIME_INFINITE</code> no timeout.

Returns

The operation status.

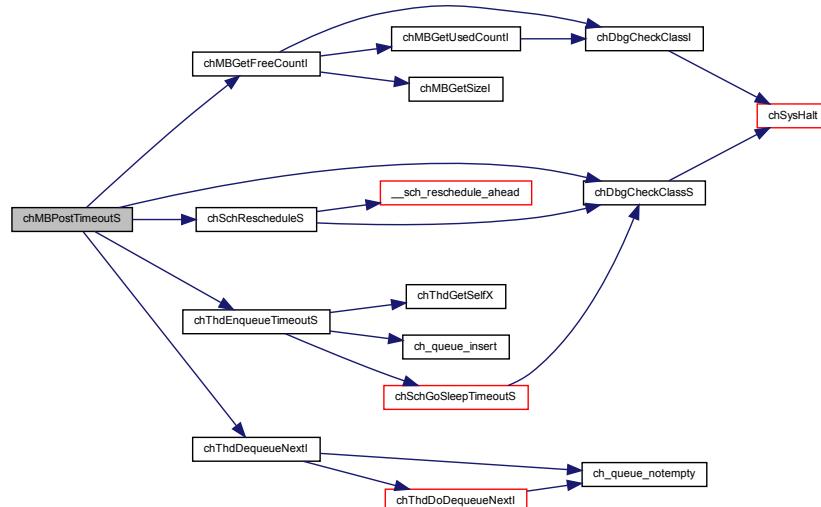
Return values

<i>MSG_OK</i>	if a message has been correctly posted.
<i>MSG_RESET</i>	if the mailbox has been reset.
<i>MSG_TIMEOUT</i>	if the operation has timed out.

Function Class:

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

Here is the call graph for this function:

**7.41.3.6 chMBPostI()**

```
msg_t chMBPostI (
    mailbox_t * mbp,
    msg_t msg )
```

Posts a message into a mailbox.

This variant is non-blocking, the function returns a timeout condition if the queue is full.

Parameters

in	<i>mbp</i>	the pointer to an initialized <code>mailbox_t</code> object
in	<i>msg</i>	the message to be posted on the mailbox

Returns

The operation status.

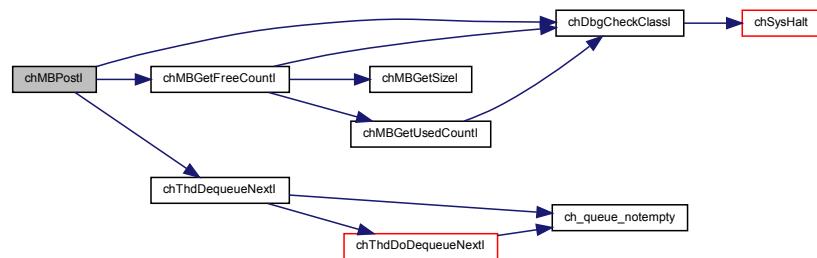
Return values

<i>MSG_OK</i>	if a message has been correctly posted.
<i>MSG_RESET</i>	if the mailbox has been reset.
<i>MSG_TIMEOUT</i>	if the mailbox is full and the message cannot be posted.

Function Class:

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

Here is the call graph for this function:

**7.41.3.7 chMBPostAheadTimeout()**

```
msg_t chMBPostAheadTimeout (
    mailbox_t * mbp,
    msg_t msg,
    sysinterval_t timeout )
```

Posts an high priority message into a mailbox.

The invoking thread waits until a empty slot in the mailbox becomes available or the specified time runs out.

Parameters

in	<i>mbp</i>	the pointer to an initialized <code>mailbox_t</code> object
in	<i>msg</i>	the message to be posted on the mailbox
in	<i>timeout</i>	the number of ticks before the operation timeouts, the following special values are allowed: <ul style="list-style-type: none"> • <code>TIME_IMMEDIATE</code> immediate timeout. • <code>TIME_INFINITE</code> no timeout.

Returns

The operation status.

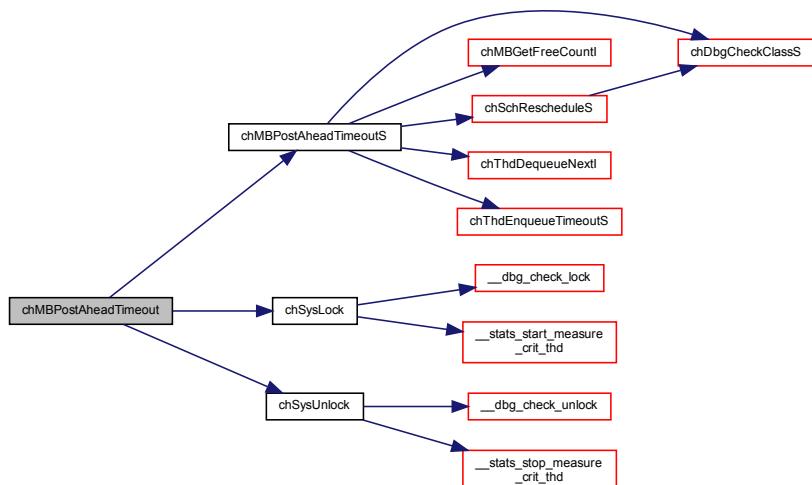
Return values

<i>MSG_OK</i>	if a message has been correctly posted.
<i>MSG_RESET</i>	if the mailbox has been reset.
<i>MSG_TIMEOUT</i>	if the operation has timed out.

Function Class:

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

Here is the call graph for this function:

**7.41.3.8 chMBPostAheadTimeoutS()**

```
msg_t chMBPostAheadTimeoutS (
    mailbox_t * mbp,
    msg_t msg,
    sysinterval_t timeout )
```

Posts an high priority message into a mailbox.

The invoking thread waits until a empty slot in the mailbox becomes available or the specified time runs out.

Parameters

in	<i>mbp</i>	the pointer to an initialized <code>mailbox_t</code> object
in	<i>msg</i>	the message to be posted on the mailbox
in	<i>timeout</i>	the number of ticks before the operation timeouts, the following special values are allowed: • <i>TIME_IMMEDIATE</i> immediate timeout. • <i>TIME_INFINITE</i> no timeout.

Returns

The operation status.

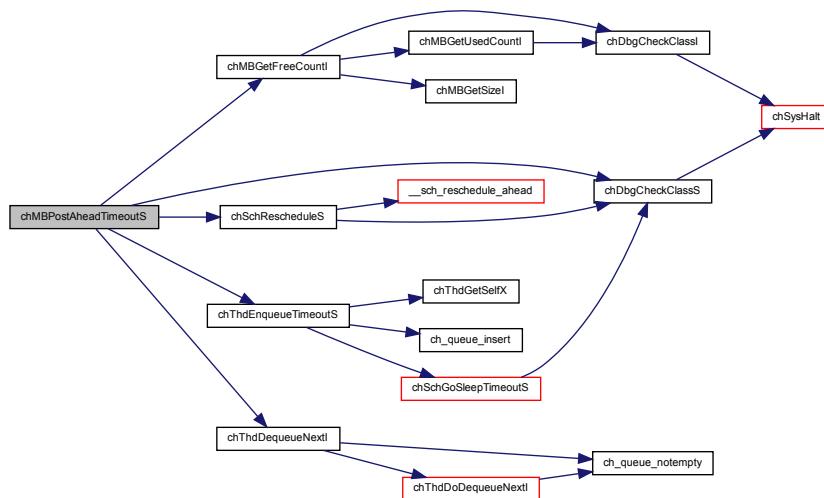
Return values

MSG_OK	if a message has been correctly posted.
MSG_RESET	if the mailbox has been reset.
MSG_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:



7.41.3.9 chMBPostAhead()

```
msg_t chMBPostAheadI ( mailbox_t * mbp,  
                         msg_t msg )
```

Posts an high priority message into a mailbox.

This variant is non-blocking, the function returns a timeout condition if the queue is full.

Parameters

in	<i>mbp</i>	the pointer to an initialized <code>mailbox_t</code> object
in	<i>msg</i>	the message to be posted on the mailbox

Returns

The operation status.

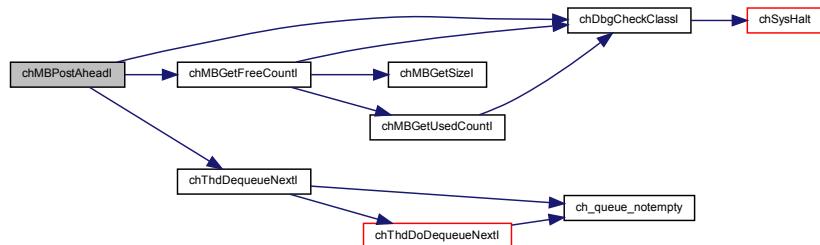
Return values

<i>MSG_OK</i>	if a message has been correctly posted.
<i>MSG_RESET</i>	if the mailbox has been reset.
<i>MSG_TIMEOUT</i>	if the mailbox is full and the message cannot be posted.

Function Class:

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

Here is the call graph for this function:

**7.41.3.10 chMBFetchTimeout()**

```
msg_t chMBFetchTimeout (
    mailbox_t * mbp,
    msg_t * msgp,
    sysinterval_t timeout )
```

Retrieves a message from a mailbox.

The invoking thread waits until a message is posted in the mailbox or the specified time runs out.

Parameters

in	<i>mbp</i>	the pointer to an initialized <code>mailbox_t</code> object
out	<i>msgp</i>	pointer to a message variable for the received message
in	<i>timeout</i>	the number of ticks before the operation timeouts, the following special values are allowed: <ul style="list-style-type: none"> • <code>TIME_IMMEDIATE</code> immediate timeout. • <code>TIME_INFINITE</code> no timeout.

Returns

The operation status.

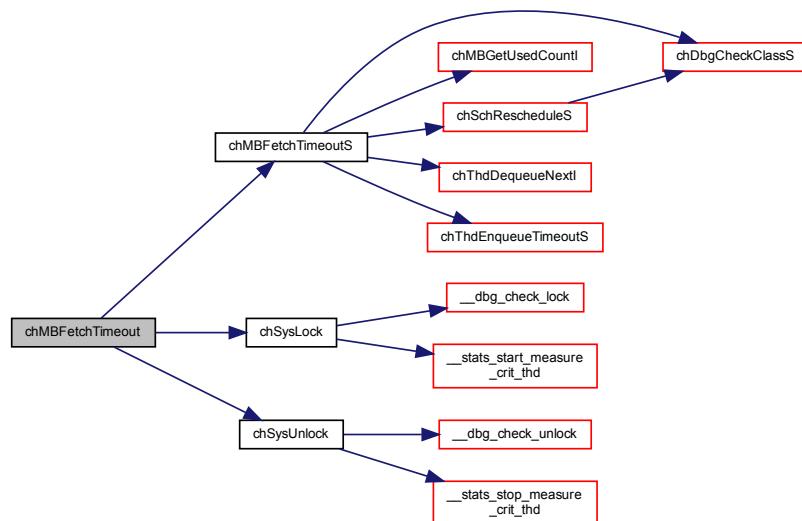
Return values

<i>MSG_OK</i>	if a message has been correctly fetched.
<i>MSG_RESET</i>	if the mailbox has been reset.
<i>MSG_TIMEOUT</i>	if the operation has timed out.

Function Class:

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

Here is the call graph for this function:

**7.41.3.11 chMBFetchTimeoutS()**

```

msg_t chMBFetchTimeoutS (
    mailbox_t * mbp,
    msg_t * msgp,
    sysinterval_t timeout )
  
```

Retrieves a message from a mailbox.

The invoking thread waits until a message is posted in the mailbox or the specified time runs out.

Parameters

in	<i>mbp</i>	the pointer to an initialized <code>mailbox_t</code> object
out	<i>msgp</i>	pointer to a message variable for the received message
in	<i>timeout</i>	the number of ticks before the operation timeouts, the following special values are allowed: <ul style="list-style-type: none"> <code>TIME_IMMEDIATE</code> immediate timeout. <code>TIME_INFINITE</code> no timeout.

Returns

The operation status.

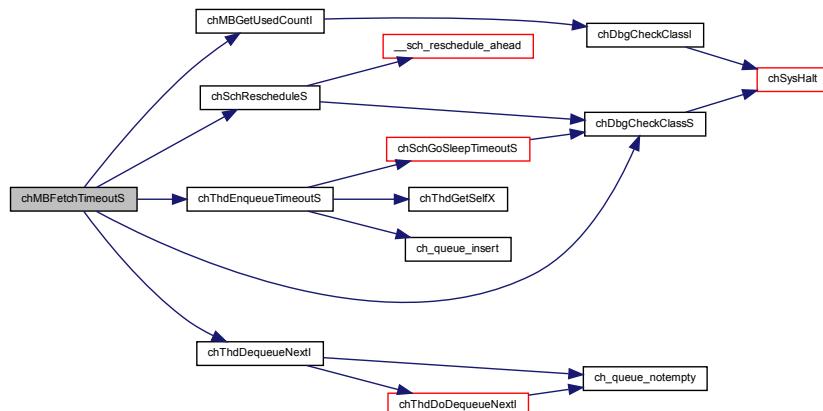
Return values

<code>MSG_OK</code>	if a message has been correctly fetched.
<code>MSG_RESET</code>	if the mailbox has been reset.
<code>MSG_TIMEOUT</code>	if the operation has timed out.

Function Class:

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

Here is the call graph for this function:



7.41.3.12 `chMBFetchI()`

```
msg_t chMBFetchI (
    mailbox_t * mbp,
    msg_t * msgp )
```

Retrieves a message from a mailbox.

This variant is non-blocking, the function returns a timeout condition if the queue is empty.

Parameters

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

Returns

The operation status.

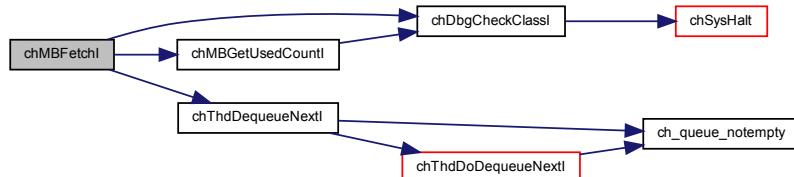
Return values

<code>MSG_OK</code>	if a message has been correctly fetched.
<code>MSG_RESET</code>	if the mailbox has been reset.
<code>MSG_TIMEOUT</code>	if the mailbox is empty and a message cannot be fetched.

Function Class:

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

Here is the call graph for this function:



7.41.3.13 chMBGetSizeI()

```
static size_t chMBGetSizeI (
    const mailbox_t * mbp) [inline], [static]
```

Returns the mailbox buffer size as number of messages.

Parameters

in	<i>mbp</i>	the pointer to an initialized <code>mailbox_t</code> object
----	------------	---

Returns

The size of the mailbox.

Function Class:

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

7.41.3.14 chMBGetUsedCountI()

```
static size_t chMBGetUsedCountI (
    const mailbox_t * mbp ) [inline], [static]
```

Returns the number of used message slots into a mailbox.

Parameters

in	<i>mbp</i>	the pointer to an initialized mailbox_t object
----	------------	--

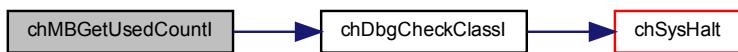
Returns

The number of queued messages.

Function Class:

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

Here is the call graph for this function:

**7.41.3.15 chMBGetFreeCountI()**

```
static size_t chMBGetFreeCountI (
    const mailbox_t * mbp ) [inline], [static]
```

Returns the number of free message slots into a mailbox.

Parameters

in	<i>mbp</i>	the pointer to an initialized mailbox_t object
----	------------	--

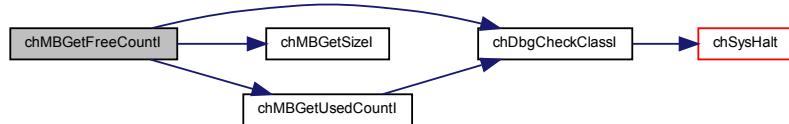
Returns

The number of empty message slots.

Function Class:

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

Here is the call graph for this function:

**7.41.3.16 chMBPeekI()**

```
static msg_t chMBPeekI (
    const mailbox_t * mbp ) [inline], [static]
```

Returns the next message in the queue without removing it.

Precondition

A message must be waiting in the queue for this function to work or it would return garbage. The correct way to use this macro is to use [chMBGetUsedCountI\(\)](#) and then use this macro, all within a lock state.

Parameters

in	<i>mbp</i>	the pointer to an initialized mailbox_t object
----	------------	--

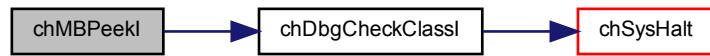
Returns

The next message in queue.

Function Class:

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

Here is the call graph for this function:

**7.41.3.17 chMBResumeX()**

```
static void chMBResumeX (
    mailbox_t * mbp ) [inline], [static]
```

Terminates the reset state.

Parameters

in	<i>mbp</i>	the pointer to an initialized <code>mailbox_t</code> object
----	------------	---

Function Class:

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

7.42 Pipes

7.42.1 Detailed Description

Macros

- `#define __PIPE_DATA(name, buffer, size)`
Data part of a static pipe initializer.
- `#define PIPE_DECL(name, buffer, size) pipe_t name = __PIPE_DATA(name, buffer, size)`
Static pipe initializer.

Data Structures

- `struct pipe_t`
Structure representing a pipe object.

Functions

- `static size_t pipe_write (pipe_t *pp, const uint8_t *bp, size_t n)`
Non-blocking pipe write.
- `static size_t pipe_read (pipe_t *pp, uint8_t *bp, size_t n)`
Non-blocking pipe read.
- `void chPipeObjectInit (pipe_t *pp, uint8_t *buf, size_t n)`
Initializes a mailbox_t object.
- `void chPipeReset (pipe_t *pp)`
Resets a pipe_t object.
- `size_t chPipeWriteTimeout (pipe_t *pp, const uint8_t *bp, size_t n, sysinterval_t timeout)`
Pipe write with timeout.
- `size_t chPipeReadTimeout (pipe_t *pp, uint8_t *bp, size_t n, sysinterval_t timeout)`
Pipe read with timeout.
- `static size_t chPipeGetSize (const pipe_t *pp)`
Returns the pipe buffer size as number of bytes.
- `static size_t chPipeGetUsedCount (const pipe_t *pp)`
Returns the number of used byte slots into a pipe.
- `static size_t chPipeGetFreeCount (const pipe_t *pp)`
Returns the number of free byte slots into a pipe.
- `static void chPipeResume (pipe_t *pp)`
Terminates the reset state.

7.42.2 Macro Definition Documentation

7.42.2.1 __PIPE_DATA

```
#define __PIPE_DATA(
    name,
    buffer,
    size )
```

Value:

```
{
    (uint8_t *) (buffer),
    (uint8_t *) (buffer) + size,
    (uint8_t *) (buffer),
    (uint8_t *) (buffer),
    (size_t) 0,
    false,
    NULL,
    NULL,
    __MUTEX_DATA(name.cmtx),
    __MUTEX_DATA(name.wmtx),
    __MUTEX_DATA(name.rmtx),
}
```

Data part of a static pipe initializer.

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

Parameters

in	<i>name</i>	the name of the pipe variable
in	<i>buffer</i>	pointer to the pipe buffer array of <code>uint8_t</code>
in	<i>size</i>	number of <code>uint8_t</code> elements in the buffer array

7.42.2.2 PIPE_DECL

```
#define PIPE_DECL(
    name,
    buffer,
    size ) pipe_t name = __PIPE_DATA(name, buffer, size)
```

Static pipe initializer.

Statically initialized pipes require no explicit initialization using `chPipeObjectInit()`.

Parameters

in	<i>name</i>	the name of the pipe variable
in	<i>buffer</i>	pointer to the pipe buffer array of <code>uint8_t</code>
in	<i>size</i>	number of <code>uint8_t</code> elements in the buffer array

7.42.3 Function Documentation

7.42.3.1 pipe_write()

```
static size_t pipe_write (
    pipe_t * pp,
    const uint8_t * bp,
    size_t n ) [static]
```

Non-blocking pipe write.

The function writes data from a buffer to a pipe. The operation completes when the specified amount of data has been transferred or when the pipe buffer has been filled.

Parameters

in	<i>pp</i>	the pointer to an initialized pipe_t object
in	<i>bp</i>	pointer to the data buffer
in	<i>n</i>	the maximum amount of data to be transferred, the value 0 is reserved

Returns

The number of bytes effectively transferred.

Function Class:

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

7.42.3.2 pipe_read()

```
static size_t pipe_read (
    pipe_t * pp,
    uint8_t * bp,
    size_t n ) [static]
```

Non-blocking pipe read.

The function reads data from a pipe into a buffer. The operation completes when the specified amount of data has been transferred or when the pipe buffer has been emptied.

Parameters

in	<i>pp</i>	the pointer to an initialized pipe_t object
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

Returns

The number of bytes effectively transferred.

Function Class:

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

7.42.3.3 chPipeObjectInit()

```
void chPipeObjectInit (
    pipe_t * pp,
    uint8_t * buf,
    size_t n )
```

Initializes a [mailbox_t](#) object.

Parameters

out	<i>pp</i>	the pointer to the pipe_t structure to be initialized
in	<i>buf</i>	pointer to the pipe buffer as an array of uint8_t
in	<i>n</i>	number of elements in the buffer array

Function Class:

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

7.42.3.4 chPipeReset()

```
void chPipeReset (
    pipe_t * pp )
```

Resets a [pipe_t](#) object.

All the waiting threads are resumed with status [MSG_RESET](#) and the queued data is lost.

Postcondition

The pipe is in reset state, all operations will fail and return [MSG_RESET](#) until the mailbox is enabled again using [chPipeResumeX\(\)](#).

Parameters

in	<i>pp</i>	the pointer to an initialized pipe_t object
----	-----------	---

Function Class:

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

7.42.3.5 chPipeWriteTimeout()

```
size_t chPipeWriteTimeout (
    pipe_t * pp,
    const uint8_t * bp,
    size_t n,
    sysinterval_t timeout )
```

Pipe write with timeout.

The function writes data from a buffer to a pipe. The operation completes when the specified amount of data has been transferred or after the specified timeout or if the pipe has been reset.

Parameters

in	<i>pp</i>	the pointer to an initialized <code>pipe_t</code> object
in	<i>bp</i>	pointer to the data buffer
in	<i>n</i>	the number of bytes to be written, the value 0 is reserved
in	<i>timeout</i>	the number of ticks before the operation timeouts, the following special values are allowed: <ul style="list-style-type: none"> • <code>TIME_IMMEDIATE</code> immediate timeout. • <code>TIME_INFINITE</code> no timeout.

Returns

The number of bytes effectively transferred. A number lower than *n* means that a timeout occurred or the pipe went in reset state.

Function Class:

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

7.42.3.6 chPipeReadTimeout()

```
size_t chPipeReadTimeout (
    pipe_t * pp,
    uint8_t * bp,
    size_t n,
    sysinterval_t timeout )
```

Pipe read with timeout.

The function reads data from a pipe into a buffer. The operation completes when the specified amount of data has been transferred or after the specified timeout or if the pipe has been reset.

Parameters

in	<i>pp</i>	the pointer to an initialized <code>pipe_t</code> object
out	<i>bp</i>	pointer to the data buffer
in	<i>n</i>	the number of bytes to be read, the value 0 is reserved
Chibios/RT	<i>timeout</i>	the number of ticks before the operation timeouts, the following special values are allowed: <ul style="list-style-type: none"> • <code>TIME_IMMEDIATE</code> immediate timeout. • <code>TIME_INFINITE</code> no timeout.

Returns

The number of bytes effectively transferred. A number lower than `n` means that a timeout occurred or the pipe went in reset state.

Function Class:

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

7.42.3.7 chPipeGetSize()

```
static size_t chPipeGetSize (
    const pipe_t * pp ) [inline], [static]
```

Returns the pipe buffer size as number of bytes.

Parameters

in	<code>pp</code>	the pointer to an initialized <code>pipe_t</code> object
----	-----------------	--

Returns

The size of the pipe.

Function Class:

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

7.42.3.8 chPipeGetUsedCount()

```
static size_t chPipeGetUsedCount (
    const pipe_t * pp ) [inline], [static]
```

Returns the number of used byte slots into a pipe.

Parameters

in	<code>pp</code>	the pointer to an initialized <code>pipe_t</code> object
----	-----------------	--

Returns

The number of queued bytes.

Function Class:

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

7.42.3.9 chPipeGetFreeCount()

```
static size_t chPipeGetFreeCount (
    const pipe_t * pp ) [inline], [static]
```

Returns the number of free byte slots into a pipe.

Parameters

in	pp	the pointer to an initialized pipe_t object
----	----	---

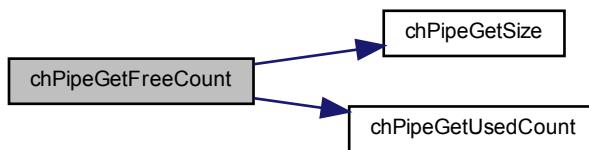
Returns

The number of empty byte slots.

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:

**7.42.3.10 chPipeResume()**

```
static void chPipeResume (
    pipe_t * pp ) [inline], [static]
```

Terminates the reset state.

Parameters

in	<i>pp</i>	the pointer to an initialized pipe_t object
----	-----------	---

Function Class:

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

7.43 Delegate Threads

7.43.1 Detailed Description

Typedefs

- `typedef msg_t(* delegate_veneer_t) (va_list *argsp)`
Type of a delegate veneer function.
- `typedef msg_t(* delegate_fn0_t) (void)`
Type of a delegate function with no parameters.
- `typedef msg_t(* delegate_fn1_t) (msg_t p1)`
Type of a delegate function with one parameter.
- `typedef msg_t(* delegate_fn2_t) (msg_t p1, msg_t p2)`
Type of a delegate function with two parameters.
- `typedef msg_t(* delegate_fn3_t) (msg_t p1, msg_t p2, msg_t p3)`
Type of a delegate function with three parameters.
- `typedef msg_t(* delegate_fn4_t) (msg_t p1, msg_t p2, msg_t p3, msg_t p4)`
Type of a delegate function with four parameters.

Functions

- `msg_t __ch_delegate_fn0 (va_list *argsp)`
Veneer for functions with no parameters.
- `msg_t __ch_delegate_fn1 (va_list *argsp)`
Veneer for functions with one parameter.
- `msg_t __ch_delegate_fn2 (va_list *argsp)`
Veneer for functions with two parameters.
- `msg_t __ch_delegate_fn3 (va_list *argsp)`
Veneer for functions with three parameters.
- `msg_t __ch_delegate_fn4 (va_list *argsp)`
Veneer for functions with four parameters.
- `msg_t chDelegateCallVeneer (thread_t *tp, delegate_veneer_t veneer,...)`
Triggers a function call on a delegate thread.
- `void chDelegateDispatch (void)`
Call messages dispatching.
- `msg_t chDelegateDispatchTimeout (sysinterval_t timeout)`
Call messages dispatching with timeout.
- `static msg_t chDelegateCallDirect0 (thread_t *tp, delegate_fn0_t func)`
Direct call to a function with no parameters.
- `static msg_t chDelegateCallDirect1 (thread_t *tp, delegate_fn1_t func, msg_t p1)`
Direct call to a function with one parameter.
- `static msg_t chDelegateCallDirect2 (thread_t *tp, delegate_fn2_t func, msg_t p1, msg_t p2)`
Direct call to a function with two parameters.
- `static msg_t chDelegateCallDirect3 (thread_t *tp, delegate_fn3_t func, msg_t p1, msg_t p2, msg_t p3)`
Direct call to a function with three parameters.
- `static msg_t chDelegateCallDirect4 (thread_t *tp, delegate_fn4_t func, msg_t p1, msg_t p2, msg_t p3, msg_t p4)`
Direct call to a function with four parameters.

7.43.2 Typedef Documentation

7.43.2.1 `delegate_veneer_t`

```
typedef msg_t (* delegate_veneer_t) (va_list *argsp)
```

Type of a delegate veneer function.

7.43.2.2 `delegate_fn0_t`

```
typedef msg_t (* delegate_fn0_t) (void)
```

Type of a delegate function with no parameters.

7.43.2.3 `delegate_fn1_t`

```
typedef msg_t (* delegate_fn1_t) (msg_t p1)
```

Type of a delegate function with one parameter.

7.43.2.4 `delegate_fn2_t`

```
typedef msg_t (* delegate_fn2_t) (msg_t p1, msg_t p2)
```

Type of a delegate function with two parameters.

7.43.2.5 `delegate_fn3_t`

```
typedef msg_t (* delegate_fn3_t) (msg_t p1, msg_t p2, msg_t p3)
```

Type of a delegate function with three parameters.

7.43.2.6 `delegate_fn4_t`

```
typedef msg_t (* delegate_fn4_t) (msg_t p1, msg_t p2, msg_t p3, msg_t p4)
```

Type of a delegate function with four parameters.

7.43.3 Function Documentation

7.43.3.1 `__ch_delegate_fn0()`

```
msg_t __ch_delegate_fn0 (
    va_list * args)
```

Veneer for functions with no parameters.

Parameters

in	<i>argsp</i>	the list of arguments
----	--------------	-----------------------

Returns

The function return value.

7.43.3.2 __ch_delegate_fn1()

```
msg_t __ch_delegate_fn1 (
    va_list * argsp )
```

Veneer for functions with one parameter.

Parameters

in	<i>argsp</i>	the list of arguments
----	--------------	-----------------------

Returns

The function return value.

7.43.3.3 __ch_delegate_fn2()

```
msg_t __ch_delegate_fn2 (
    va_list * argsp )
```

Veneer for functions with two parameters.

Parameters

in	<i>argsp</i>	the list of arguments
----	--------------	-----------------------

Returns

The function return value.

7.43.3.4 __ch_delegate_fn3()

```
msg_t __ch_delegate_fn3 (
    va_list * argsp )
```

Veneer for functions with three parameters.

Parameters

in	<i>argsp</i>	the list of arguments
----	--------------	-----------------------

Returns

The function return value.

7.43.3.5 __ch_delegate_fn4()

```
msg_t __ch_delegate_fn4 (
    va_list * argsp )
```

Veneer for functions with four parameters.

Parameters

in	<i>argsp</i>	the list of arguments
----	--------------	-----------------------

Returns

The function return value.

7.43.3.6 chDelegateCallVeneer()

```
msg_t chDelegateCallVeneer (
    thread_t * tp,
    delegate_veneer_t veneer,
    ... )
```

Triggers a function call on a delegate thread.

Note

The thread must be executing `chDelegateDispatchTimeout()` in order to have the functions called.

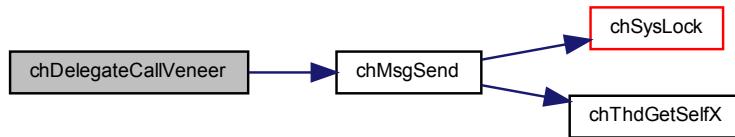
Parameters

in	<i>tp</i>	pointer to the delegate thread
in	<i>vee</i> neer	pointer to the veneer function to be called
in	...	variable number of parameters

Returns

The function return value casted to msg_t. It is garbage for functions returning void.

Here is the call graph for this function:

**7.43.3.7 chDelegateDispatch()**

```
void chDelegateDispatch (
    void )
```

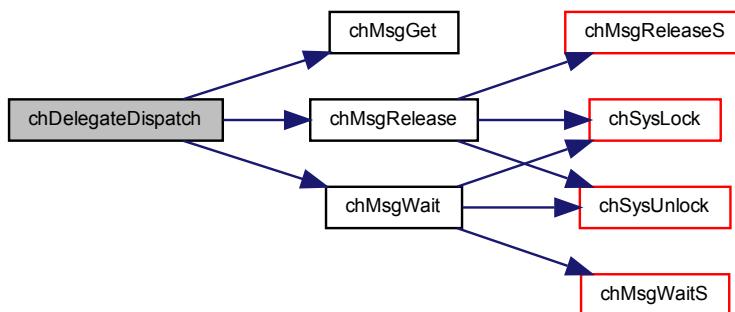
Call messages dispatching.

The function awaits for an incoming call messages and calls the specified functions, then it returns. In case multiple threads are sending messages then the requests are served in priority order.

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:



7.43.3.8 chDelegateDispatchTimeout()

```
msg_t chDelegateDispatchTimeout (
    sysinterval_t timeout )
```

Call messages dispatching with timeout.

The function awaits for an incoming call messages and calls the specified functions, then it returns. In case multiple threads are sending messages then the requests are served in priority order.

Parameters

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 function outcome.

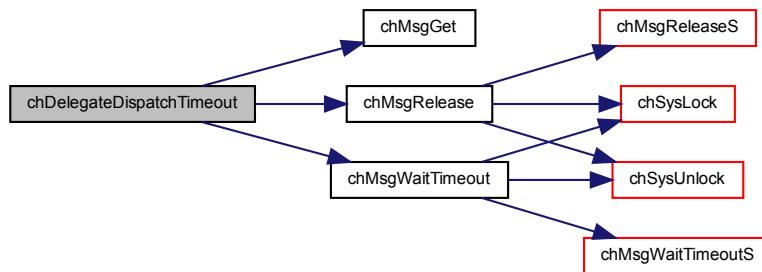
Return values

<i>MSG_OK</i>	if a function has been called.
<i>MSG_TIMEOUT</i>	if a timeout occurred.

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:



7.43.3.9 chDelegateCallDirect0()

```
static msg_t chDelegateCallDirect0 (
    thread_t * tp,
    delegate_fn0_t func ) [inline], [static]
```

Direct call to a function with no parameters.

Note

The return value is assumed to be not larger than a data pointer type. If you need a portable function then use [chDelegateCallVeneer\(\)](#) instead.

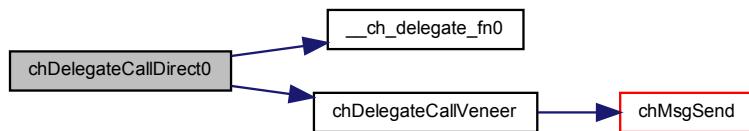
Parameters

in	<i>tp</i>	pointer to the delegate thread
in	<i>func</i>	pointer to the function to be called

Returns

The function return value as a `msg_t`.

Here is the call graph for this function:



7.43.3.10 chDelegateCallDirect1()

```
static msg_t chDelegateCallDirect1 (
    thread_t * tp,
    delegate_fn1_t func,
    msg_t p1 ) [inline], [static]
```

Direct call to a function with one parameter.

Note

The return value and parameters are assumed to be not larger than a data pointer type. If you need a portable function then use [chDelegateCallVeneer\(\)](#) instead.

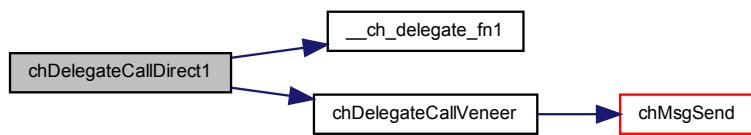
Parameters

in	<i>tp</i>	pointer to the delegate thread
in	<i>func</i>	pointer to the function to be called
in	<i>p1</i>	parameter 1 passed as a <code>msg_t</code>

Returns

The function return value as a `msg_t`.

Here is the call graph for this function:



7.43.3.11 chDelegateCallDirect2()

```
static msg_t chDelegateCallDirect2 (
    thread_t * tp,
    delegate_fn2_t func,
    msg_t p1,
    msg_t p2 ) [inline], [static]
```

Direct call to a function with two parameters.

Note

The return value and parameters are assumed to be not larger than a data pointer type. If you need a portable function then use [chDelegateCallVeneer\(\)](#) instead.

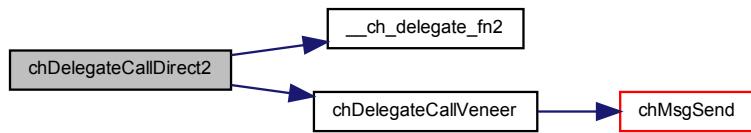
Parameters

in	<i>tp</i>	pointer to the delegate thread
in	<i>func</i>	pointer to the function to be called
in	<i>p1</i>	parameter 1 passed as a <code>msg_t</code>
in	<i>p2</i>	parameter 2 passed as a <code>msg_t</code>

Returns

The function return value as a `msg_t`.

Here is the call graph for this function:

**7.43.3.12 chDelegateCallDirect3()**

```
static msg_t chDelegateCallDirect3 (
    thread_t * tp,
    delegate_fn3_t func,
    msg_t p1,
    msg_t p2,
    msg_t p3 ) [inline], [static]
```

Direct call to a function with three parameters.

Note

The return value and parameters are assumed to be not larger than a data pointer type. If you need a portable function then use [chDelegateCallVeneer\(\)](#) instead.

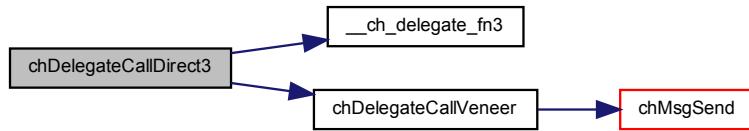
Parameters

in	<i>tp</i>	pointer to the delegate thread
in	<i>func</i>	pointer to the function to be called
in	<i>p1</i>	parameter 1 passed as a <code>msg_t</code>
in	<i>p2</i>	parameter 2 passed as a <code>msg_t</code>
in	<i>p3</i>	parameter 3 passed as a <code>msg_t</code>

Returns

The function return value as a `msg_t`.

Here is the call graph for this function:

**7.43.3.13 chDelegateCallDirect4()**

```
static msg_t chDelegateCallDirect4 (
    thread_t * tp,
    delegate_fn4_t func,
    msg_t p1,
    msg_t p2,
    msg_t p3,
    msg_t p4 ) [inline], [static]
```

Direct call to a function with four parameters.

Note

The return value and parameters are assumed to be not larger than a data pointer type. If you need a portable function then use [chDelegateCallVeneer\(\)](#) instead.

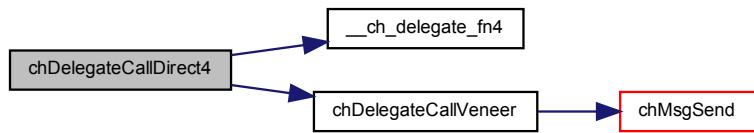
Parameters

in	<i>tp</i>	pointer to the delegate thread
in	<i>func</i>	pointer to the function to be called
in	<i>p1</i>	parameter 1 passed as a <code>msg_t</code>
in	<i>p2</i>	parameter 2 passed as a <code>msg_t</code>
in	<i>p3</i>	parameter 3 passed as a <code>msg_t</code>
in	<i>p4</i>	parameter 4 passed as a <code>msg_t</code>

Returns

The function return value as a `msg_t`.

Here is the call graph for this function:



7.44 Jobs Queues

7.44.1 Detailed Description

Macros

- #define `MSG_JOB_NULL` ((`msg_t`)-2)
Dispatcher return code in case of a `JOB_NUL` has been received.

Typedefs

- typedef struct `ch_jobs_queue` `jobs_queue_t`
Type of a jobs queue.
- typedef void(* `job_function_t`) (void *arg)
Type of a job function.
- typedef struct `ch_job_descriptor` `job_descriptor_t`
Type of a job descriptor.

Data Structures

- struct `ch_jobs_queue`
Type of a jobs queue.
- struct `ch_job_descriptor`
Type of a job descriptor.

Functions

- static void `chJobObjectInit` (`jobs_queue_t` *`jqp`, `size_t` `jobsn`, `job_descriptor_t` *`jobsbuf`, `msg_t` *`msgbuf`)
Initializes a jobs queue object.
- static `job_descriptor_t` * `chJobGet` (`jobs_queue_t` *`jqp`)
Allocates a free job object.
- static `job_descriptor_t` * `chJobGet1` (`jobs_queue_t` *`jqp`)
Allocates a free job object.
- static `job_descriptor_t` * `chJobGetTimeoutS` (`jobs_queue_t` *`jqp`, `sysinterval_t` `timeout`)
Allocates a free job object.
- static `job_descriptor_t` * `chJobGetTimeout` (`jobs_queue_t` *`jqp`, `sysinterval_t` `timeout`)
Allocates a free job object.
- static void `chJobPost1` (`jobs_queue_t` *`jqp`, `job_descriptor_t` *`jp`)
Posts a job object.
- static void `chJobPostS` (`jobs_queue_t` *`jqp`, `job_descriptor_t` *`jp`)
Posts a job object.
- static void `chJobPost` (`jobs_queue_t` *`jqp`, `job_descriptor_t` *`jp`)
Posts a job object.
- static void `chJobPostAhead1` (`jobs_queue_t` *`jqp`, `job_descriptor_t` *`jp`)
Posts an high priority job object.
- static void `chJobPostAheadS` (`jobs_queue_t` *`jqp`, `job_descriptor_t` *`jp`)
Posts an high priority job object.
- static void `chJobPostAhead` (`jobs_queue_t` *`jqp`, `job_descriptor_t` *`jp`)
Posts an high priority job object.
- static `msg_t` `chJobDispatch` (`jobs_queue_t` *`jqp`)
Waits for a job then executes it.
- static `msg_t` `chJobDispatchTimeout` (`jobs_queue_t` *`jqp`, `sysinterval_t` `timeout`)
Waits for a job then executes it.

7.44.2 Macro Definition Documentation

7.44.2.1 MSG_JOB_NULL

```
#define MSG_JOB_NULL ( (msg_t) -2)
```

Dispatcher return code in case of a JOB_NUL has been received.

7.44.3 Typedef Documentation

7.44.3.1 jobs_queue_t

```
typedef struct ch_jobs_queue jobs_queue_t
```

Type of a jobs queue.

7.44.3.2 job_function_t

```
typedef void(* job_function_t) (void *arg)
```

Type of a job function.

7.44.3.3 job_descriptor_t

```
typedef struct ch_job_descriptor job_descriptor_t
```

Type of a job descriptor.

7.44.4 Function Documentation

7.44.4.1 chJobObjectInit()

```
static void chJobObjectInit (
    jobs_queue_t * jqp,
    size_t jobsn,
    job_descriptor_t * jobsbuf,
    msg_t * msgbuf ) [inline], [static]
```

Initializes a jobs queue object.

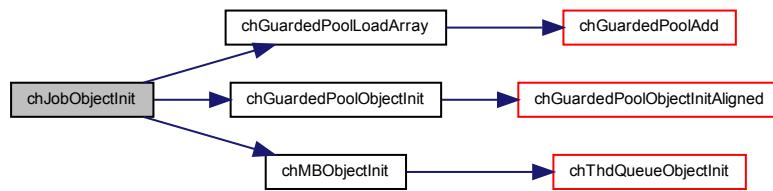
Parameters

out	<i>jqp</i>	pointer to a <code>jobs_queue_t</code> structure
in	<i>jobsn</i>	number of jobs available
in	<i>jobsbuf</i>	pointer to the buffer of jobs, it must be able to hold <code>jobsn job_descriptor_t</code> structures
in	<i>msgbuf</i>	pointer to the buffer of messages, it must be able to hold <code>jobsn msg_t</code> messages

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:



7.44.4.2 chJobGet()

```
static job_descriptor_t* chJobGet (
    jobs_queue_t * jqp ) [inline], [static]
```

Allocates a free job object.

Parameters

in	<i>jqp</i>	pointer to a <code>jobs_queue_t</code> structure
----	------------	--

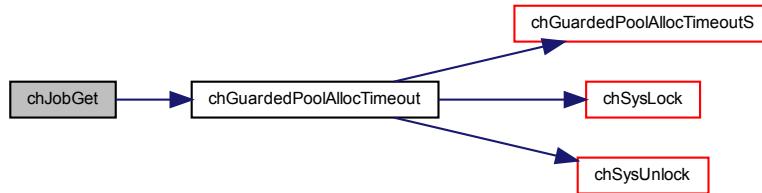
Returns

The pointer to the allocated job 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:

**7.44.4.3 chJobGetI()**

```
static job_descriptor_t* chJobGetI (
    jobs_queue_t * jqp ) [inline], [static]
```

Allocates a free job object.

Parameters

in	<i>jqp</i>	pointer to a <code>jobs_queue_t</code> structure
----	------------	--

Returns

The pointer to the allocated job object.

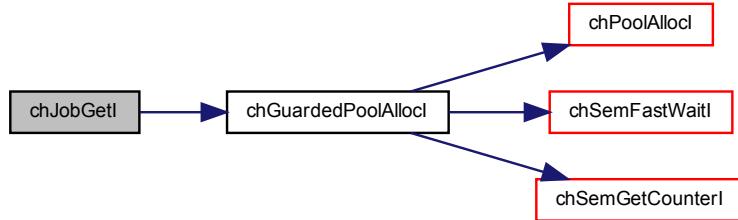
Return values

<code>NULL</code>	if a job object is not immediately available.
-------------------	---

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:

**7.44.4.4 chJobGetTimeoutS()**

```
static job_descriptor_t* chJobGetTimeouts (
    jobs_queue_t * jqp,
    sysinterval_t timeout ) [inline], [static]
```

Allocates a free job object.

Parameters

in	<i>jqp</i>	pointer to a <code>jobs_queue_t</code> structure
in	<i>timeout</i>	the number of ticks before the operation timeouts, the following special values are allowed: <ul style="list-style-type: none"> • <code>TIME_IMMEDIATE</code> immediate timeout. • <code>TIME_INFINITE</code> no timeout.

Returns

The pointer to the allocated job object.

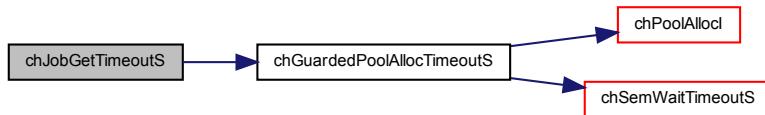
Return values

<code>NULL</code>	if a job object is not available 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:

**7.44.4.5 chJobGetTimeout()**

```
static job_descriptor_t* chJobGetTimeout (
    jobs_queue_t * jqp,
    sysinterval_t timeout ) [inline], [static]
```

Allocates a free job object.

Parameters

in	<i>jqp</i>	pointer to a <code>jobs_queue_t</code> structure
in	<i>timeout</i>	the number of ticks before the operation timeouts, the following special values are allowed: <ul style="list-style-type: none"> • <code>TIME_IMMEDIATE</code> immediate timeout. • <code>TIME_INFINITE</code> no timeout.

Returns

The pointer to the allocated job object.

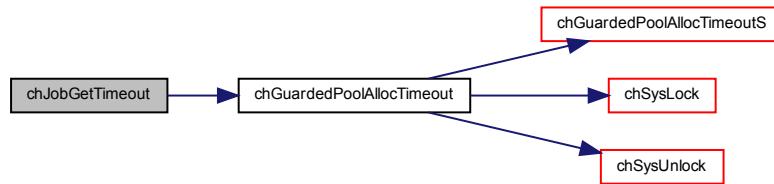
Return values

NULL	if a job object is not available 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:

**7.44.4.6 chJobPostI()**

```
static void chJobPostI (
    jobs_queue_t * jqp,
    job_descriptor_t * jp ) [inline], [static]
```

Posts a job object.

Note

By design the object can be always immediately posted.

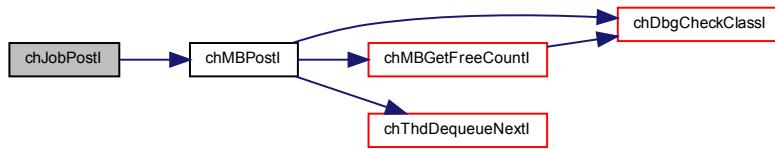
Parameters

in	<i>jqp</i>	pointer to a <code>jobs_queue_t</code> structure
in	<i>jp</i>	pointer to the job object to 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:

**7.44.4.7 chJobPostS()**

```
static void chJobPosts (
    jobs_queue_t * jqp,
    job_descriptor_t * jp ) [inline], [static]
```

Posts a job object.

Note

By design the object can be always immediately posted.

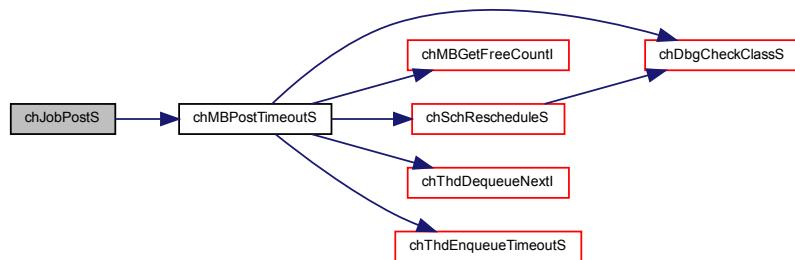
Parameters

in	<i>jqp</i>	pointer to a <code>jobs_queue_t</code> structure
in	<i>jp</i>	pointer to the job object to be posted

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:

**7.44.4.8 chJobPost()**

```
static void chJobPost (
    jobs_queue_t * jqp,
    job_descriptor_t * jp ) [inline], [static]
```

Posts a job object.

Note

By design the object can be always immediately posted.

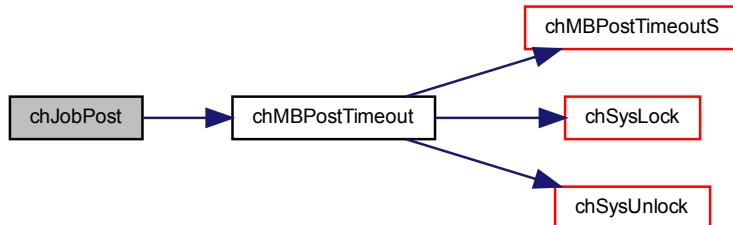
Parameters

in	<i>jqp</i>	pointer to a <code>jobs_queue_t</code> structure
in	<i>jp</i>	pointer to the job object to be posted

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:

**7.44.4.9 chJobPostAheadI()**

```
static void chJobPostAheadI (
    jobs_queue_t * jqp,
    job_descriptor_t * jp ) [inline], [static]
```

Posts an high priority job object.

Note

By design the object can be always immediately posted.

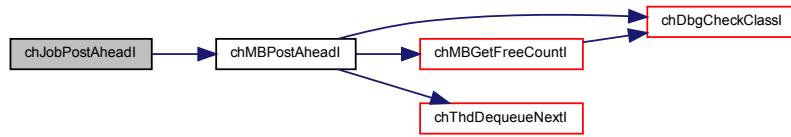
Parameters

in	<i>jqp</i>	pointer to a <code>jobs_queue_t</code> structure
in	<i>jp</i>	pointer to the job object to 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:

**7.44.4.10 chJobPostAheadS()**

```
static void chJobPostAheadS (
    jobs_queue_t * jqp,
    job_descriptor_t * jp ) [inline], [static]
```

Posts an high priority job object.

Note

By design the object can be always immediately posted.

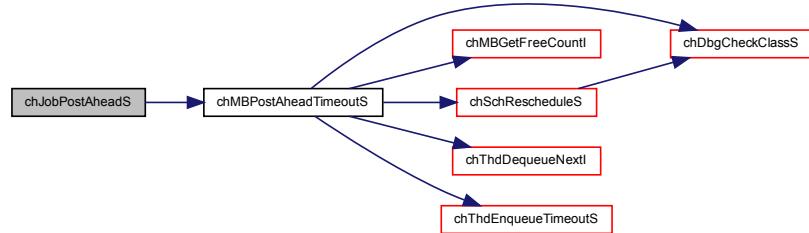
Parameters

in	<i>jqp</i>	pointer to a <code>jobs_queue_t</code> structure
in	<i>jp</i>	pointer to the job object to be posted

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:



7.44.4.11 chJobPostAhead()

```

static void chJobPostAhead (
    jobs_queue_t * jqp,
    job_descriptor_t * jp ) [inline], [static]
  
```

Posts an high priority job object.

Note

By design the object can be always immediately posted.

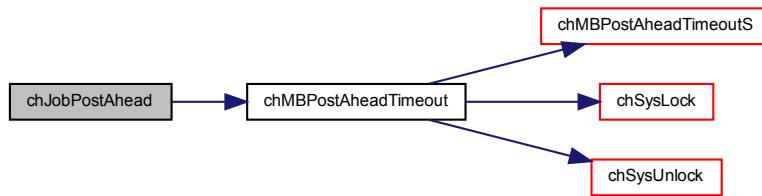
Parameters

in	<i>jqp</i>	pointer to a <code>jobs_queue_t</code> structure
in	<i>jp</i>	pointer to the job object to be posted

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:

**7.44.4.12 chJobDispatch()**

```
static msg_t chJobDispatch (
    jobs_queue_t * jqp ) [inline], [static]
```

Waits for a job then executes it.

Parameters

in	<i>jqp</i>	pointer to a <code>jobs_queue_t</code> structure
----	------------	--

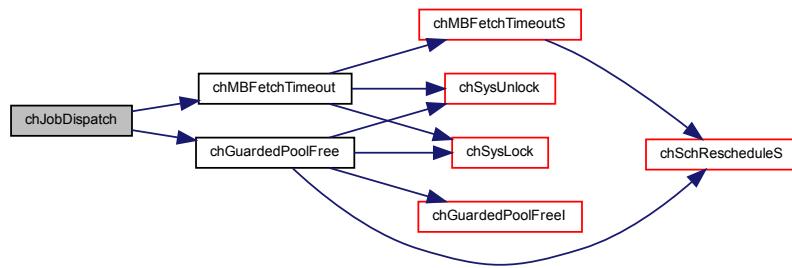
Returns

The function outcome.

Return values

<code>MSG_OK</code>	if a job has been executed.
<code>MSG_RESET</code>	if the internal mailbox has been reset.
<code>MSG_JOB_NULL</code>	if a <code>JOB_NULL</code> has been received.

Here is the call graph for this function:



7.44.4.13 `chJobDispatchTimeout()`

```
static msg_t chJobDispatchTimeout (
    jobs_queue_t * jqp,
    sysinterval_t timeout ) [inline], [static]
```

Waits for a job then executes it.

Parameters

in	<i>jqp</i>	pointer to a <code>jobs_queue_t</code> structure
in	<i>timeout</i>	the number of ticks before the operation timeouts, the following special values are allowed: <ul style="list-style-type: none"> • <code>TIME_IMMEDIATE</code> immediate timeout. • <code>TIME_INFINITE</code> no timeout.

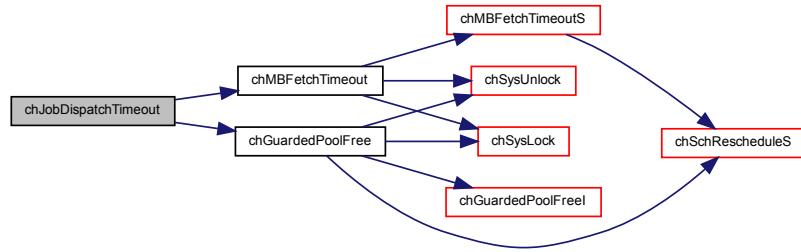
Returns

The function outcome.

Return values

<code>MSG_OK</code>	if a job has been executed.
<code>MSG_TIMEOUT</code>	if a timeout occurred.
<code>MSG_RESET</code>	if the internal mailbox has been reset.
<code>MSG_JOB_NULL</code>	if a <code>JOB_NULL</code> has been received.

Here is the call graph for this function:



7.45 Memory Management

7.45.1 Detailed Description

Memory Management services.

Modules

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

7.46 Core Memory Manager

7.46.1 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 [Memory Heaps](#)).
- Memory pools allocator (see [Memory Pools](#)).

By having a centralized memory provider the various allocators can coexist and share the main memory.
This allocator, alone, is also useful for very simple applications that just require a simple way to get memory blocks.

Precondition

In order to use the core memory manager APIs the CH_CFG_USE_MEMCORE option must be enabled in [chconf.h](#).

Note

Compatible with RT and NIL.

Macros

- `#define CH_CFG_MEMCORE_SIZE 0`
Managed RAM size.
- `#define chCoreAllocAlignedWithOffsetI chCoreAllocFromTopI`
Allocates a memory block.
- `#define chCoreAllocAlignedWithOffset chCoreAllocFromTop`
Allocates a memory block.

Typedefs

- `typedef void *(* memgetfunc_t) (size_t size, unsigned align)`
Memory get function.
- `typedef void *(* memgetfunc2_t) (size_t size, unsigned align, size_t offset)`
Enhanced memory get function.

Data Structures

- `struct memcore_t`
Type of memory core object.

Functions

- void `__core_init` (void)

Low level memory manager initialization.
- void * `chCoreAllocFromBaseI` (size_t size, unsigned align, size_t offset)

Allocates a memory block starting from the lowest address upward.
- void * `chCoreAllocFromTopI` (size_t size, unsigned align, size_t offset)

Allocates a memory block starting from the top address downward.
- void * `chCoreAllocFromBase` (size_t size, unsigned align, size_t offset)

Allocates a memory block starting from the lowest address upward.
- void * `chCoreAllocFromTop` (size_t size, unsigned align, size_t offset)

Allocates a memory block starting from the top address downward.
- size_t `chCoreGetStatusX` (void)

Core memory status.
- static void * `chCoreAllocAlignedI` (size_t size, unsigned align)

Allocates a memory block.
- static void * `chCoreAllocAligned` (size_t size, unsigned align)

Allocates a memory block.
- static void * `chCoreAllocI` (size_t size)

Allocates a memory block.
- static void * `chCoreAlloc` (size_t size)

Allocates a memory block.

Variables

- `memcore_t ch_memcore`

Memory core descriptor.

7.46.2 Macro Definition Documentation

7.46.2.1 CH_CFG_MEMCORE_SIZE

```
#define CH_CFG_MEMCORE_SIZE 0
```

Managed RAM size.

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

Note

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

Requires `CH_CFG_USE_MEMCORE`.

7.46.2.2 chCoreAllocAlignedWithOffsetI

```
#define chCoreAllocAlignedWithOffsetI chCoreAllocFromTopI
```

Allocates a memory block.

Note

This is a generic form with unspecified allocation position.

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.

7.46.2.3 chCoreAllocAlignedWithOffset

```
#define chCoreAllocAlignedWithOffset chCoreAllocFromTop
```

Allocates a memory block.

Note

This is a generic form with unspecified allocation position.

Function Class:

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

7.46.3 Typedef Documentation

7.46.3.1 memgetfunc_t

```
typedef void*(* memgetfunc_t) (size_t size, unsigned align)
```

Memory get function.

7.46.3.2 memgetfunc2_t

```
typedef void*(* memgetfunc2_t) (size_t size, unsigned align, size_t offset)
```

Enhanced memory get function.

7.46.4 Function Documentation

7.46.4.1 __core_init()

```
void __core_init (
    void )
```

Low level memory manager initialization.

Function Class:

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

7.46.4.2 chCoreAllocFromBaseI()

```
void * chCoreAllocFromBaseI (
    size_t size,
    unsigned align,
    size_t offset )
```

Allocates a memory block starting from the lowest address upward.

This function allocates a block of `offset + size` bytes. The returned pointer has `offset` bytes before its address and `size` bytes after.

Parameters

in	<code>size</code>	the size of the block to be allocated.
in	<code>align</code>	desired memory alignment
in	<code>offset</code>	aligned pointer offset

Returns

A pointer to the allocated memory block.

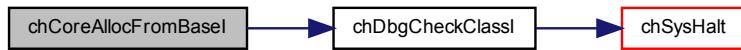
Return values

<code>NULL</code>	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:



7.46.4.3 chCoreAllocFromTopI()

```
void * chCoreAllocFromTopI (
    size_t size,
    unsigned align,
    size_t offset )
```

Allocates a memory block starting from the top address downward.

This function allocates a block of `offset + size` bytes. The returned pointer has `offset` bytes before its address and `size` bytes after.

Parameters

in	<code>size</code>	the size of the block to be allocated.
in	<code>align</code>	desired memory alignment
in	<code>offset</code>	aligned pointer offset

Returns

A pointer to the allocated memory block.

Return values

<code>NULL</code>	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:



7.46.4.4 chCoreAllocFromBase()

```
void * chCoreAllocFromBase (
    size_t size,
    unsigned align,
    size_t offset )
```

Allocates a memory block starting from the lowest address upward.

This function allocates a block of `offset + size` bytes. The returned pointer has `offset` bytes before its address and `size` bytes after.

Parameters

in	<i>size</i>	the size of the block to be allocated.
in	<i>align</i>	desired memory alignment
in	<i>offset</i>	aligned pointer offset

Returns

A pointer to the allocated memory block.

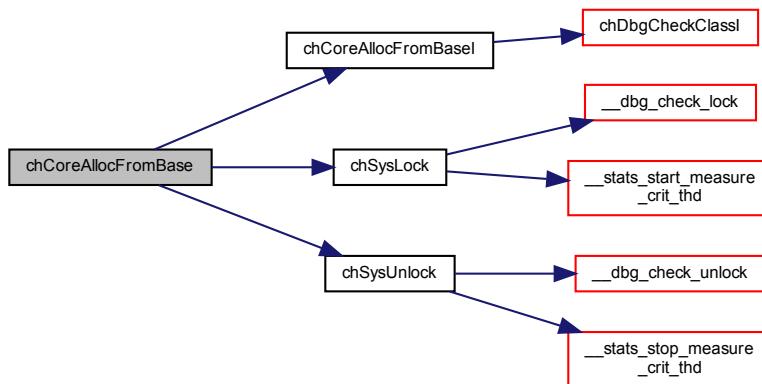
Return values

<i>NULL</i>	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:

**7.46.4.5 chCoreAllocFromTop()**

```
void * chCoreAllocFromTop (
    size_t size,
    unsigned align,
    size_t offset )
```

Allocates a memory block starting from the top address downward.

This function allocates a block of `offset + size` bytes. The returned pointer has `offset` bytes before its address and `size` bytes after.

Parameters

in	<code>size</code>	the size of the block to be allocated.
in	<code>align</code>	desired memory alignment
in	<code>offset</code>	aligned pointer offset

Returns

A pointer to the allocated memory block.

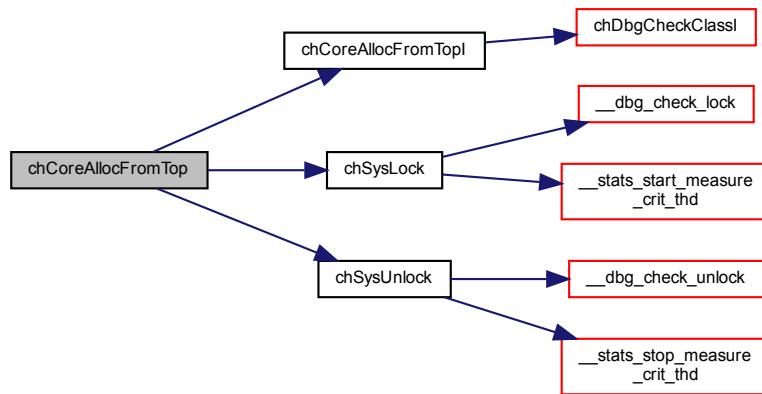
Return values

<code>NULL</code>	allocation failed, core memory exhausted.
-------------------	---

Function Class:

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

Here is the call graph for this function:

**7.46.4.6 chCoreGetStatusX()**

```
size_t chCoreGetStatusX (
    void )
```

Core memory status.

Returns

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

Function Class:

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

7.46.4.7 chCoreAllocAlignedI()

```
static void* chCoreAllocAlignedI (
    size_t size,
    unsigned align ) [inline], [static]
```

Allocates a memory block.

The allocated block is guaranteed to be properly aligned to the specified alignment.

Note

This is a generic form with unspecified allocation position.

Parameters

in	<i>size</i>	the size of the block to be allocated.
in	<i>align</i>	desired memory alignment

Returns

A pointer to the allocated memory block.

Return values

<i>NULL</i>	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.

7.46.4.8 chCoreAllocAligned()

```
static void* chCoreAllocAligned (
    size_t size,
    unsigned align) [inline], [static]
```

Allocates a memory block.

The allocated block is guaranteed to be properly aligned to the specified alignment.

Note

This is a generic form with unspecified allocation position.

Parameters

in	<i>size</i>	the size of the block to be allocated
in	<i>align</i>	desired memory alignment

Returns

A pointer to the allocated memory block.

Return values

<i>NULL</i>	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.

7.46.4.9 chCoreAllocI()

```
static void* chCoreAllocI (
    size_t size ) [inline], [static]
```

Allocates a memory block.

The allocated block is guaranteed to be properly aligned for a pointer data type.

Note

This is a generic form with unspecified allocation position.

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.

7.46.4.10 chCoreAlloc()

```
static void* chCoreAlloc (
    size_t size ) [inline], [static]
```

Allocates a memory block.

The allocated block is guaranteed to be properly aligned for a pointer data type.

Note

This is a generic form with unspecified allocation position.

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.

7.46.5 Variable Documentation

7.46.5.1 ch_memcore

`memcore_t ch_memcore`

Memory core descriptor.

7.47 Memory Heaps

7.47.1 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 and there is the ability to return memory blocks aligned to arbitrary powers of two.

Precondition

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

Note

Compatible with RT and NIL.

Macros

- `#define CH_HEAP_ALIGNMENT 8U`
Minimum alignment used for heap.
- `#define CH_HEAP_AREA(name, size)`
Allocation of an aligned static heap buffer.

Typedefs

- `typedef struct memory_heap memory_heap_t`
Type of a memory heap.
- `typedef union heap_header heap_header_t`
Type of a memory heap header.

Data Structures

- `union heap_header`
Memory heap block header.
- `struct memory_heap`
Structure describing a memory heap.

Functions

- void `__heap_init` (void)
Initializes the default heap.
- void `chHeapObjectInit` (`memory_heap_t` *heapp, void *buf, `size_t` size)
Initializes a memory heap from a static memory area.
- void * `chHeapAllocAligned` (`memory_heap_t` *heapp, `size_t` size, `unsigned` align)
Allocates a block of memory from the heap by using the first-fit algorithm.
- void `chHeapFree` (void *p)
Frees a previously allocated memory block.
- `size_t` `chHeapStatus` (`memory_heap_t` *heapp, `size_t` *totalp, `size_t` *largestp)
Reports the heap status.
- static void * `chHeapAlloc` (`memory_heap_t` *heapp, `size_t` size)
Allocates a block of memory from the heap by using the first-fit algorithm.
- static `size_t` `chHeapGetSize` (const void *p)
Returns the size of an allocated block.

Variables

- static `memory_heap_t` `default_heap`
Default heap descriptor.

7.47.2 Macro Definition Documentation

7.47.2.1 CH_HEAP_ALIGNMENT

```
#define CH_HEAP_ALIGNMENT 8U
```

Minimum alignment used for heap.

Note

Cannot use the sizeof operator in this macro.

7.47.2.2 CH_HEAP_AREA

```
#define CH_HEAP_AREA( \
    name, \
    size )
```

Value:

```
    ALIGNED_VAR(CH_HEAP_ALIGNMENT) \
    uint8_t name[MEM_ALIGN_NEXT((size), CH_HEAP_ALIGNMENT)] \
```

Allocation of an aligned static heap buffer.

7.47.3 Typedef Documentation

7.47.3.1 memory_heap_t

```
typedef struct memory_heap memory_heap_t
```

Type of a memory heap.

7.47.3.2 heap_header_t

```
typedef union heap_header heap_header_t
```

Type of a memory heap header.

7.47.4 Function Documentation

7.47.4.1 __heap_init()

```
void __heap_init (
    void )
```

Initializes the default heap.

Function Class:

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

7.47.4.2 chHeapObjectInit()

```
void chHeapObjectInit (
    memory_heap_t * heapp,
    void * buf,
    size_t size )
```

Initializes a memory heap from a static memory area.

Note

The heap buffer base and size are adjusted if the passed buffer is not aligned to CH_HEAP_ALIGNMENT. This mean that the effective heap size can be less than `size`.

Parameters

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

Function Class:

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

7.47.4.3 chHeapAllocAligned()

```
void * chHeapAllocAligned (
    memory_heap_t * heapp,
    size_t size,
    unsigned align )
```

Allocates a block of memory from the heap by using the first-fit algorithm.

The allocated block is guaranteed to be properly aligned to the specified alignment.

Parameters

in	<i>heapp</i>	pointer to a heap descriptor or NULL in order to access the default heap.
in	<i>size</i>	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.
in	<i>align</i>	desired memory alignment

Returns

A pointer to the aligned 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.

7.47.4.4 chHeapFree()

```
void chHeapFree (
    void * p )
```

Frees a previously allocated memory block.

Parameters

in	<i>p</i>	pointer to the memory block to be freed
----	----------	---

Function Class:

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

7.47.4.5 chHeapStatus()

```
size_t chHeapStatus (
    memory_heap_t * heapp,
    size_t * totalp,
    size_t * largestp )
```

Reports the heap status.

Note

This function is meant to be used in the test suite, it should not be really useful for the application code.

Parameters

in	<i>heapp</i>	pointer to a heap descriptor or NULL in order to access the default heap.
in	<i>totalp</i>	pointer to a variable that will receive the total fragmented free space or NULL
in	<i>largestp</i>	pointer to a variable that will receive the largest free block found space or NULL

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.

7.47.4.6 chHeapAlloc()

```
static void* chHeapAlloc (
    memory_heap_t * heapp,
    size_t size ) [inline], [static]
```

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.

Parameters

in	<i>heapp</i>	pointer to a heap descriptor or <code>NULL</code> in order to access the default heap.
in	<i>size</i>	the size of the block to be allocated. Note that the allocated block may be a bit bigger than the requested size for alignment and fragmentation reasons.

Returns

A pointer to the allocated block.

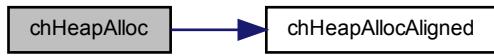
Return values

<code>NULL</code>	if the block cannot be allocated.
-------------------	-----------------------------------

Function Class:

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

Here is the call graph for this function:

**7.47.4.7 chHeapGetSize()**

```
static size_t chHeapGetSize (
    const void * p ) [inline], [static]
```

Returns the size of an allocated block.

Note

The returned value is the requested size, the real size is the same value aligned to the next CH_HEAP_ALIGNMENT multiple.

Parameters

in	<i>p</i>	pointer to the memory block
----	----------	-----------------------------

Returns

Size of the block.

Function Class:

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

7.47.5 Variable Documentation

7.47.5.1 default_heap

```
memory_heap_t default_heap [static]
```

Default heap descriptor.

7.48 Memory Pools

7.48.1 Detailed Description

Memory Pools related APIs and services.

Operation mode

The Memory Pools APIs allow to allocate/free fixed size objects in **constant time** and reliably without memory fragmentation problems.

Memory Pools do not enforce any alignment constraint on the contained object however the objects must be properly aligned to contain a pointer to void.

Precondition

In order to use the memory pools APIs the CH_CFG_USE_MEMPOOLS option must be enabled in `chconf.h`.

Note

Compatible with RT and NIL.

Macros

- `#define __MEMORYPOOL_DATA(name, size, align, provider) {NULL, size, align, provider}`
Data part of a static memory pool initializer.
- `#define MEMORYPOOL_DECL(name, size, align, provider) memory_pool_t name = __MEMORYPOOL_DATA(name, size, align, provider)`
Static memory pool initializer.
- `#define __GUARDEDMEMORYPOOL_DATA(name, size, align)`
Data part of a static guarded memory pool initializer.
- `#define GUARDEDMEMORYPOOL_DECL(name, size, align) guarded_memory_pool_t name = __GUARDEDMEMORYPOOL_DATA(name, size, align)`
Static guarded memory pool initializer.

Data Structures

- struct `pool_header`
Memory pool free object header.
- struct `memory_pool_t`
Memory pool descriptor.
- struct `guarded_memory_pool_t`
Guarded memory pool descriptor.

Functions

- void `chPoolObjectInitAligned` (`memory_pool_t` *mp, `size_t` size, `unsigned align`, `memgetfunc_t` provider)
Initializes an empty memory pool.
- void `chPoolLoadArray` (`memory_pool_t` *mp, `void *p`, `size_t n`)
Loads a memory pool with an array of static objects.
- void * `chPoolAlloc` (`memory_pool_t` *mp)
Allocates an object from a memory pool.
- void * `chPoolAlloc` (`memory_pool_t` *mp)
Allocates an object from a memory pool.
- void `chPoolFree` (`memory_pool_t` *mp, `void *objp`)
Releases an object into a memory pool.
- void `chPoolFree` (`memory_pool_t` *mp, `void *objp`)
Releases an object into a memory pool.
- void `chGuardedPoolObjectInitAligned` (`guarded_memory_pool_t` *gmp, `size_t size`, `unsigned align`)
Initializes an empty guarded memory pool.
- void `chGuardedPoolLoadArray` (`guarded_memory_pool_t` *gmp, `void *p`, `size_t n`)
Loads a guarded memory pool with an array of static objects.
- void * `chGuardedPoolAllocTimeoutS` (`guarded_memory_pool_t` *gmp, `sysinterval_t timeout`)
Allocates an object from a guarded memory pool.
- void * `chGuardedPoolAllocTimeout` (`guarded_memory_pool_t` *gmp, `sysinterval_t timeout`)
Allocates an object from a guarded memory pool.
- void `chGuardedPoolFree` (`guarded_memory_pool_t` *gmp, `void *objp`)
Releases an object into a guarded memory pool.
- static void `chPoolObjectInit` (`memory_pool_t` *mp, `size_t size`, `memgetfunc_t` provider)
Initializes an empty memory pool.
- static void `chPoolAdd` (`memory_pool_t` *mp, `void *objp`)
Adds an object to a memory pool.
- static void `chPoolAddl` (`memory_pool_t` *mp, `void *objp`)
Adds an object to a memory pool.
- static void `chGuardedPoolObjectInit` (`guarded_memory_pool_t` *gmp, `size_t size`)
Initializes an empty guarded memory pool.
- static `cnt_t` `chGuardedPoolGetCounterl` (`guarded_memory_pool_t` *gmp)
Gets the count of objects in a guarded memory pool.
- static void * `chGuardedPoolAlloc` (`guarded_memory_pool_t` *gmp)
Allocates an object from a guarded memory pool.
- static void `chGuardedPoolFree` (`guarded_memory_pool_t` *gmp, `void *objp`)
Releases an object into a guarded memory pool.
- static void `chGuardedPoolFreeS` (`guarded_memory_pool_t` *gmp, `void *objp`)
Releases an object into a guarded memory pool.
- static void `chGuardedPoolAdd` (`guarded_memory_pool_t` *gmp, `void *objp`)
Adds an object to a guarded memory pool.
- static void `chGuardedPoolAddl` (`guarded_memory_pool_t` *gmp, `void *objp`)
Adds an object to a guarded memory pool.
- static void `chGuardedPoolAddS` (`guarded_memory_pool_t` *gmp, `void *objp`)
Adds an object to a guarded memory pool.

7.48.2 Macro Definition Documentation

7.48.2.1 __MEMORYPOOL_DATA

```
#define __MEMORYPOOL_DATA(
    name,
    size,
    align,
    provider ) {NULL, size, align, 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>align</i>	required memory alignment
in	<i>provider</i>	memory provider function for the memory pool

7.48.2.2 MEMORYPOOL_DECL

```
#define MEMORYPOOL_DECL(
    name,
    size,
    align,
    provider ) memory_pool_t name = __MEMORYPOOL_DATA(name, size, align, provider)
```

Static memory pool initializer.

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>align</i>	required memory alignment
in	<i>provider</i>	memory provider function for the memory pool or <code>NULL</code> if the pool is not allowed to grow automatically

7.48.2.3 __GUARDEDMEMORYPOOL_DATA

```
#define __GUARDEDMEMORYPOOL_DATA(
    name,
    size,
    align )
```

Value:

```
{
    __SEMAPHORE_DATA(name.sem, (cnt_t)0),
    __MEMORYPOOL_DATA(NULL, size, align, NULL)
}
```

Data part of a static guarded 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>align</i>	required memory alignment

7.48.2.4 GUARDEDMEMORYPOOL_DECL

```
#define GUARDEDMEMORYPOOL_DECL(
    name,
    size,
    align ) guarded_memory_pool_t name = __GUARDEDMEMORYPOOL_DATA(name, size, align)
```

Static guarded memory pool initializer.

Statically initialized guarded memory pools require no explicit initialization using `chGuardedPoolInit()`.

Parameters

in	<i>name</i>	the name of the guarded memory pool variable
in	<i>size</i>	size of the memory pool contained objects
in	<i>align</i>	required memory alignment

7.48.3 Function Documentation**7.48.3.1 chPoolObjectInitAligned()**

```
void chPoolObjectInitAligned (
    memory_pool_t * mp,
    size_t size,
    unsigned align,
    memgetfunc_t provider )
```

Initializes an empty memory pool.

Parameters

<i>out</i>	<i>mp</i>	pointer to a <code>memory_pool_t</code> structure
<i>in</i>	<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.
<i>in</i>	<i>align</i>	required memory alignment
<i>in</i>	<i>provider</i>	memory provider function for the memory pool or <code>NULL</code> if the pool is not allowed to grow automatically

Function Class:

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

7.48.3.2 chPoolLoadArray()

```
void chPoolLoadArray (
    memory_pool_t * mp,
    void * p,
    size_t n )
```

Loads a memory pool with an array of static objects.

Precondition

The memory pool must already be initialized.

The array elements must be of the right size for the specified memory pool.

The array elements size must be a multiple of the alignment requirement for the pool.

Postcondition

The memory pool contains the elements of the input array.

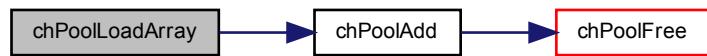
Parameters

<i>in</i>	<i>mp</i>	pointer to a <code>memory_pool_t</code> structure
<i>in</i>	<i>p</i>	pointer to the array first element
<i>in</i>	<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:



7.48.3.3 chPoolAllocI()

```
void * chPoolAllocI (
    memory_pool_t * mp )
```

Allocates an object from a memory pool.

Precondition

The memory pool must already be initialized.

Parameters

in	mp	pointer to a <code>memory_pool_t</code> structure
----	----	---

Returns

The pointer to the allocated object.

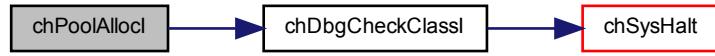
Return values

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:



7.48.3.4 chPoolAlloc()

```
void * chPoolAlloc (
    memory_pool_t * mp )
```

Allocates an object from a memory pool.

Precondition

The memory pool must already be initialized.

Parameters

in	mp	pointer to a memory_pool_t structure
----	----	--

Returns

The pointer to the allocated object.

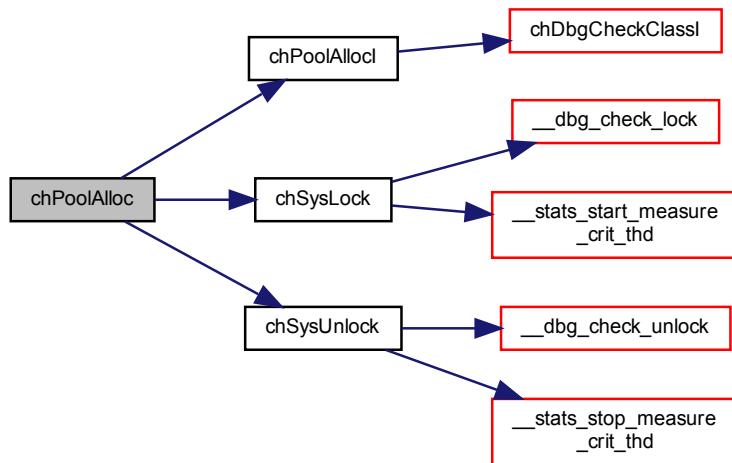
Return values

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:

**7.48.3.5 chPoolFreeI()**

```
void chPoolFreeI (
    memory_pool_t * mp,
    void * objp )
```

Releases an object into a memory pool.

Precondition

- The memory pool must already be initialized.
- The freed object must be of the right size for the specified memory pool.
- The added object must be properly aligned.

Parameters

in	<code>mp</code>	pointer to a <code>memory_pool_t</code> structure
in	<code>objp</code>	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:



7.48.3.6 chPoolFree()

```
void chPoolFree (
    memory_pool_t * mp,
    void * objp )
```

Releases an object into a memory pool.

Precondition

- The memory pool must already be initialized.
- The freed object must be of the right size for the specified memory pool.
- The added object must be properly aligned.

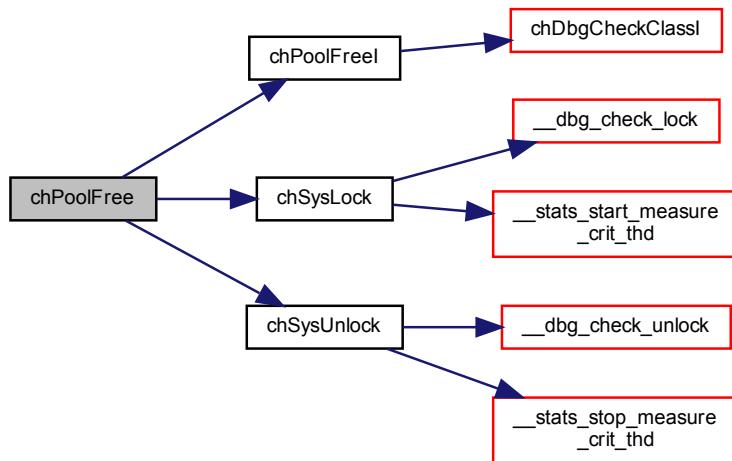
Parameters

in	<i>mp</i>	pointer to a <code>memory_pool_t</code> structure
in	<i>objp</i>	the pointer to the object to be released

Function Class:

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

Here is the call graph for this function:

**7.48.3.7 chGuardedPoolObjectInitAligned()**

```

void chGuardedPoolObjectInitAligned (
    guarded_memory_pool_t * gmp,
    size_t size,
    unsigned align )
  
```

Initializes an empty guarded memory pool.

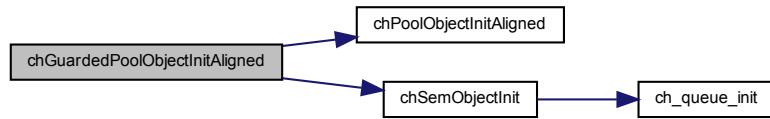
Parameters

out	<i>gmp</i>	pointer to a <code>guarded_memory_pool_t</code> structure
in	<i>size</i>	the size of the objects contained in this guarded memory pool, the minimum accepted size is the size of a pointer to void.
in	<i>align</i>	required memory alignment

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:

**7.48.3.8 chGuardedPoolLoadArray()**

```

void chGuardedPoolLoadArray (
    guarded_memory_pool_t * gmp,
    void * p,
    size_t n )
  
```

Loads a guarded memory pool with an array of static objects.

Precondition

The guarded memory pool must already be initialized.

The array elements must be of the right size for the specified guarded memory pool.

Postcondition

The guarded memory pool contains the elements of the input array.

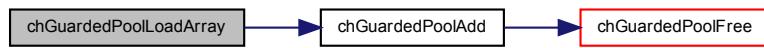
Parameters

in	<i>gmp</i>	pointer to a guarded_memory_pool_t structure
in	<i>p</i>	pointer to the array first element
in	<i>n</i>	number of elements in the array

Function Class:

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

Here is the call graph for this function:



7.48.3.9 chGuardedPoolAllocTimeoutS()

```
void * chGuardedPoolAllocTimeoutS (
    guarded_memory_pool_t * gmp,
    sysinterval_t timeout )
```

Allocates an object from a guarded memory pool.

Precondition

The guarded memory pool must already be initialized.

Parameters

in	<i>gmp</i>	pointer to a <code>guarded_memory_pool_t</code> structure
in	<i>timeout</i>	the number of ticks before the operation timeouts, the following special values are allowed: <ul style="list-style-type: none"> • <code>TIME_IMMEDIATE</code> immediate timeout. • <code>TIME_INFINITE</code> no timeout.

Returns

The pointer to the allocated object.

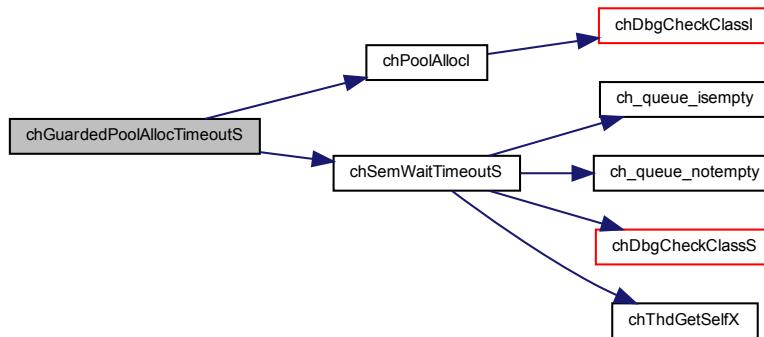
Return values

<code>NULL</code>	if the operation timed out.
-------------------	-----------------------------

Function Class:

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

Here is the call graph for this function:



7.48.3.10 `chGuardedPoolAllocTimeout()`

```
void * chGuardedPoolAllocTimeout (
    guarded_memory_pool_t * gmp,
    sysinterval_t timeout )
```

Allocates an object from a guarded memory pool.

Precondition

The guarded memory pool must already be initialized.

Parameters

in	<i>gmp</i>	pointer to a <code>guarded_memory_pool_t</code> structure
in	<i>timeout</i>	the number of ticks before the operation timeouts, the following special values are allowed: <ul style="list-style-type: none"> • <code>TIME_IMMEDIATE</code> immediate timeout. • <code>TIME_INFINITE</code> no timeout.

Returns

The pointer to the allocated object.

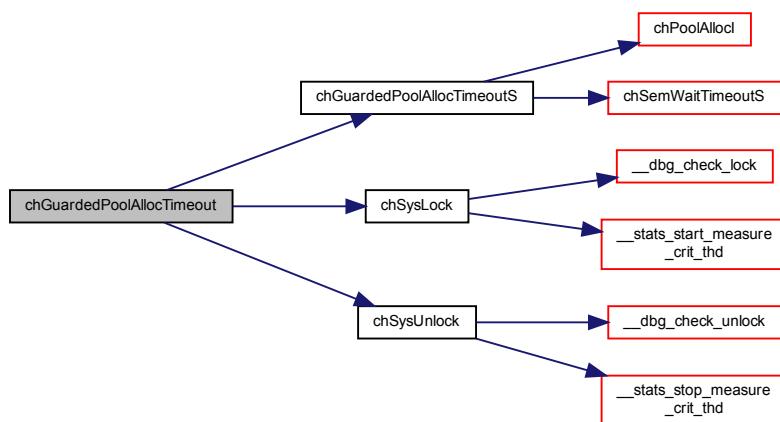
Return values

<code>NULL</code>	if the operation 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:

**7.48.3.11 chGuardedPoolFree()**

```
void chGuardedPoolFree (
    guarded_memory_pool_t * gmp,
    void * objp )
```

Releases an object into a guarded memory pool.

Precondition

The guarded memory pool must already be initialized.

The freed object must be of the right size for the specified guarded memory pool.

The added object must be properly aligned.

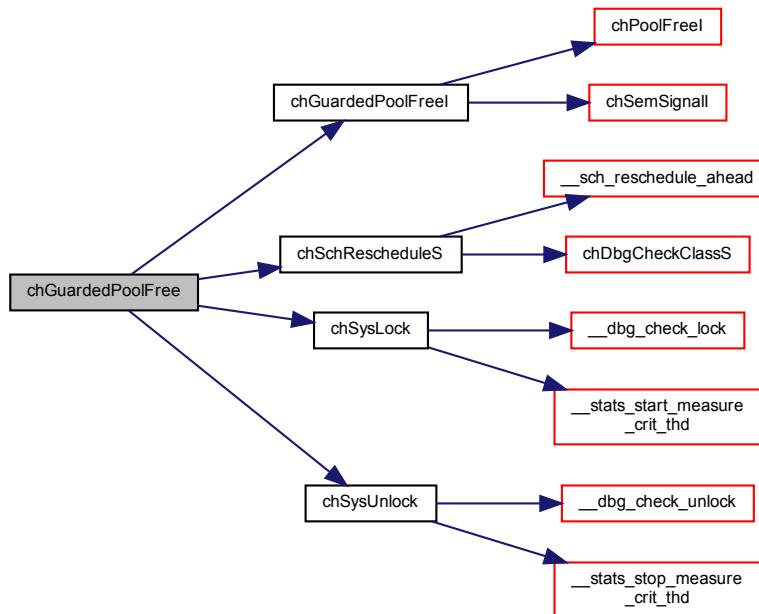
Parameters

in	<code>gmp</code>	pointer to a <code>guarded_memory_pool_t</code> structure
in	<code>objp</code>	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:

7.48.3.12 `chPoolObjectInit()`

```

static void chPoolObjectInit (
    memory_pool_t * mp,
    size_t size,
    memgetfunc_t provider ) [inline], [static]
  
```

Initializes an empty memory pool.

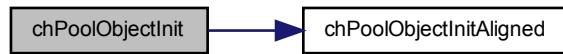
Parameters

out	<code>mp</code>	pointer to a <code>memory_pool_t</code> structure
in	<code>size</code>	the size of the objects contained in this memory pool, the minimum accepted size is the size of a pointer to void.
in	<code>provider</code>	memory provider function for the memory pool or <code>NULL</code> if the pool is not allowed to grow automatically

Function Class:

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

Here is the call graph for this function:



7.48.3.13 chPoolAdd()

```
static void chPoolAdd (
    memory_pool_t * mp,
    void * objp ) [inline], [static]
```

Adds an object to a memory pool.

Precondition

The memory pool must be already been initialized.

The added object must be of the right size for the specified memory pool.

The added object must be properly aligned.

Note

This function is just an alias for [chPoolFree\(\)](#) and has been added for clarity.

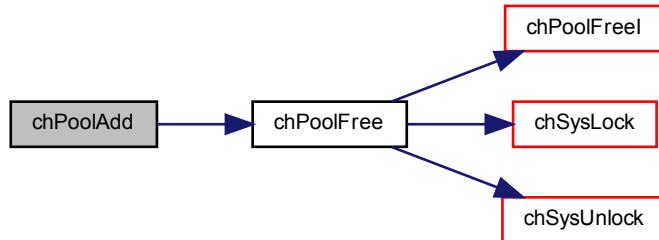
Parameters

in	<i>mp</i>	pointer to a memory_pool_t structure
in	<i>objp</i>	the pointer to the object to be added

Function Class:

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

Here is the call graph for this function:

**7.48.3.14 chPoolAddI()**

```
static void chPoolAddI (
    memory_pool_t * mp,
    void * objp ) [inline], [static]
```

Adds an object to a memory pool.

Precondition

- The memory pool must be already been initialized.
- The added object must be of the right size for the specified memory pool.
- The added object must be properly aligned.

Note

This function is just an alias for [chPoolFreeI\(\)](#) and has been added for clarity.

Parameters

in	<i>mp</i>	pointer to a <code>memory_pool_t</code> structure
in	<i>objp</i>	the pointer to the object to be 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:



7.48.3.15 chGuardedPoolObjectInit()

```
static void chGuardedPoolObjectInit (
    guarded_memory_pool_t * gmp,
    size_t size ) [inline], [static]
```

Initializes an empty guarded memory pool.

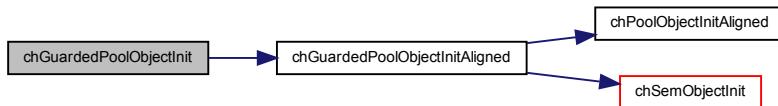
Parameters

out	<i>gmp</i>	pointer to a <code>guarded_memory_pool_t</code> structure
in	<i>size</i>	the size of the objects contained in this guarded memory pool, the minimum accepted size is the size of a pointer to void.

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:



7.48.3.16 chGuardedPoolGetCounterI()

```
static cnt_t chGuardedPoolGetCounterI (
    guarded_memory_pool_t * gmp ) [inline], [static]
```

Gets the count of objects in a guarded memory pool.

Precondition

The guarded memory pool must be already been initialized.

Parameters

in	<i>gmp</i>	pointer to a guarded_memory_pool_t structure
----	------------	--

Returns

The counter of the guard semaphore.

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:

**7.48.3.17 chGuardedPoolAllocI()**

```
static void* chGuardedPoolAllocI (
    guarded\_memory\_pool\_t * gmp) [inline], [static]
```

Allocates an object from a guarded memory pool.

Precondition

The guarded memory pool must be already been initialized.

Parameters

in	<i>gmp</i>	pointer to a guarded_memory_pool_t structure
----	------------	--

Returns

The pointer to the allocated object.

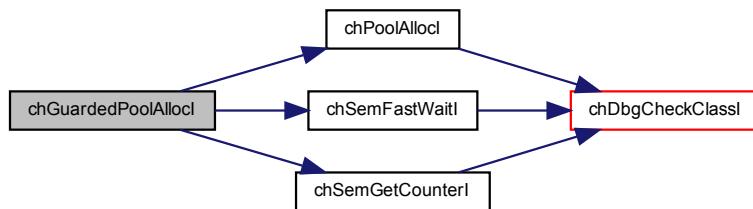
Return values

<code>NULL</code>	if the 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:

**7.48.3.18 chGuardedPoolFreeI()**

```
static void chGuardedPoolFreeI (
    guarded_memory_pool_t * gmp,
    void * objp ) [inline], [static]
```

Releases an object into a guarded memory pool.

Precondition

- The guarded memory pool must already be initialized.
- The freed object must be of the right size for the specified guarded memory pool.
- The added object must be properly aligned.

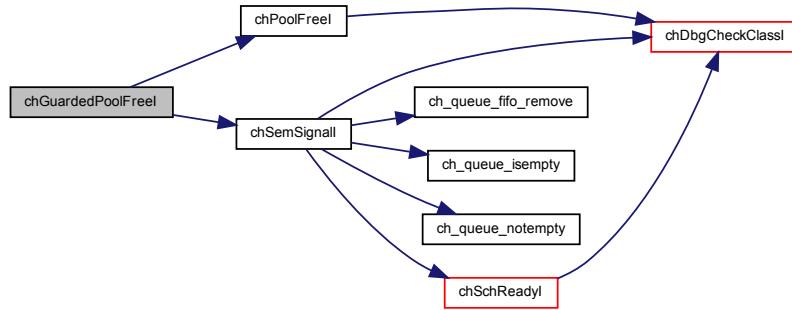
Parameters

in	<code>gmp</code>	pointer to a <code>guarded_memory_pool_t</code> structure
in	<code>objp</code>	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:



7.48.3.19 chGuardedPoolFreeS()

```
static void chGuardedPoolFreeS (
    guarded_memory_pool_t * gmp,
    void * objp ) [inline], [static]
```

Releases an object into a guarded memory pool.

Precondition

- The guarded memory pool must already be initialized.
- The freed object must be of the right size for the specified guarded memory pool.
- The added object must be properly aligned.

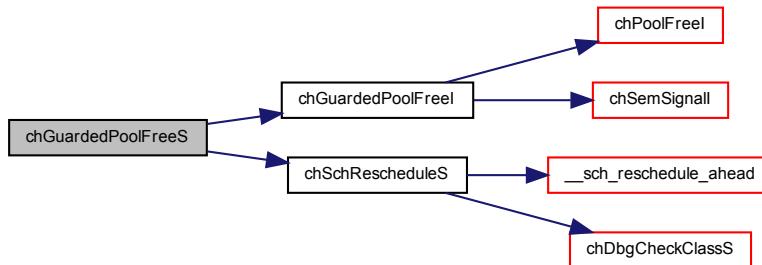
Parameters

in	<i>gmp</i>	pointer to a <code>guarded_memory_pool_t</code> structure
in	<i>objp</i>	the pointer to the object to be released

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:

**7.48.3.20 chGuardedPoolAdd()**

```
static void chGuardedPoolAdd (
    guarded_memory_pool_t * gmp,
    void * objp ) [inline], [static]
```

Adds an object to a guarded memory pool.

Precondition

- The guarded memory pool must be already been initialized.
- The added object must be of the right size for the specified guarded memory pool.
- The added object must be properly aligned.

Note

This function is just an alias for `chGuardedPoolFree()` and has been added for clarity.

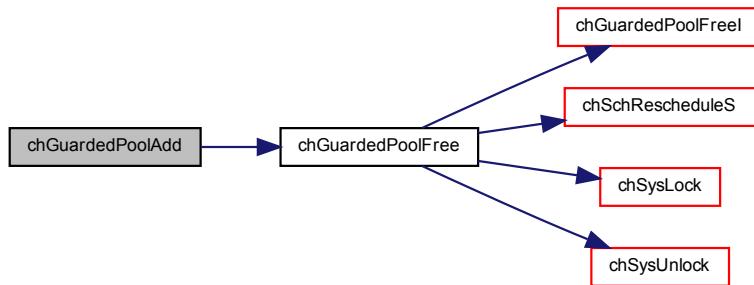
Parameters

in	<code>gmp</code>	pointer to a <code>guarded_memory_pool_t</code> structure
in	<code>objp</code>	the pointer to the object to be added

Function Class:

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

Here is the call graph for this function:

**7.48.3.21 chGuardedPoolAddI()**

```
static void chGuardedPoolAddI (
    guarded_memory_pool_t * gmp,
    void * objp ) [inline], [static]
```

Adds an object to a guarded memory pool.

Precondition

The guarded memory pool must be already been initialized.

The added object must be of the right size for the specified guarded memory pool.

The added object must be properly aligned.

Note

This function is just an alias for `chGuardedPoolFreeI()` and has been added for clarity.

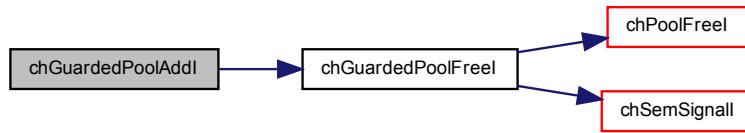
Parameters

in	<code>gmp</code>	pointer to a <code>guarded_memory_pool_t</code> structure
in	<code>objp</code>	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:



7.48.3.22 chGuardedPoolAddS()

```
static void chGuardedPoolAddS (
    guarded_memory_pool_t * gmp,
    void * objp ) [inline], [static]
```

Adds an object to a guarded memory pool.

Precondition

- The guarded memory pool must be already been initialized.
- The added object must be of the right size for the specified guarded memory pool.
- The added object must be properly aligned.

Note

This function is just an alias for [chGuardedPoolFreeI\(\)](#) and has been added for clarity.

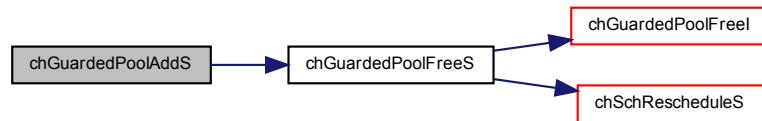
Parameters

in	<i>gmp</i>	pointer to a guarded_memory_pool_t structure
in	<i>objp</i>	the pointer to the object to be added

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:



7.49 Complex Services

7.49.1 Detailed Description

Modules

- Objects FIFOs
- Objects Caches
- Dynamic Objects Factory

7.50 Objects FIFOs

7.50.1 Detailed Description

Typedefs

- `typedef struct ch_objects_fifo objects_fifo_t`
Type of an objects FIFO.

Data Structures

- `struct ch_objects_fifo`
Type of an objects FIFO.

Functions

- `static void chFifoObjectInitAligned (objects_fifo_t *ofp, size_t objsize, size_t objn, unsigned objalign, void *objbuf, msg_t *msgbuf)`
Initializes a FIFO object.
- `static void chFifoObjectInit (objects_fifo_t *ofp, size_t objsize, size_t objn, void *objbuf, msg_t *msgbuf)`
Initializes a FIFO object.
- `static void * chFifoTakeObjectI (objects_fifo_t *ofp)`
Allocates a free object.
- `static void * chFifoTakeObjectTimeoutS (objects_fifo_t *ofp, sysinterval_t timeout)`
Allocates a free object.
- `static void * chFifoTakeObjectTimeout (objects_fifo_t *ofp, sysinterval_t timeout)`
Allocates a free object.
- `static void chFifoReturnObjectI (objects_fifo_t *ofp, void *objp)`
Releases a fetched object.
- `static void chFifoReturnObjectS (objects_fifo_t *ofp, void *objp)`
Releases a fetched object.
- `static void chFifoReturnObject (objects_fifo_t *ofp, void *objp)`
Releases a fetched object.
- `static void chFifoSendObjectI (objects_fifo_t *ofp, void *objp)`
Posts an object.
- `static void chFifoSendObjectS (objects_fifo_t *ofp, void *objp)`
Posts an object.
- `static void chFifoSendObject (objects_fifo_t *ofp, void *objp)`
Posts an object.
- `static void chFifoSendObjectAheadI (objects_fifo_t *ofp, void *objp)`
Posts an high priority object.
- `static void chFifoSendObjectAheadS (objects_fifo_t *ofp, void *objp)`
Posts an high priority object.
- `static void chFifoSendObjectAhead (objects_fifo_t *ofp, void *objp)`
Posts an high priority object.
- `static msg_t chFifoReceiveObjectI (objects_fifo_t *ofp, void **objpp)`
Fetches an object.
- `static msg_t chFifoReceiveObjectTimeoutS (objects_fifo_t *ofp, void **objpp, sysinterval_t timeout)`
Fetches an object.
- `static msg_t chFifoReceiveObjectTimeout (objects_fifo_t *ofp, void **objpp, sysinterval_t timeout)`
Fetches an object.

7.50.2 Typedef Documentation

7.50.2.1 objects_fifo_t

```
typedef struct ch_objects_fifo objects_fifo_t
```

Type of an objects FIFO.

7.50.3 Function Documentation

7.50.3.1 chFifoObjectInitAligned()

```
static void chFifoObjectInitAligned (
    objects_fifo_t * ofp,
    size_t objsize,
    size_t objn,
    unsigned objalign,
    void * objbuf,
    msg_t * msgbuf ) [inline], [static]
```

Initializes a FIFO object.

Precondition

The messages size must be a multiple of the alignment requirement.

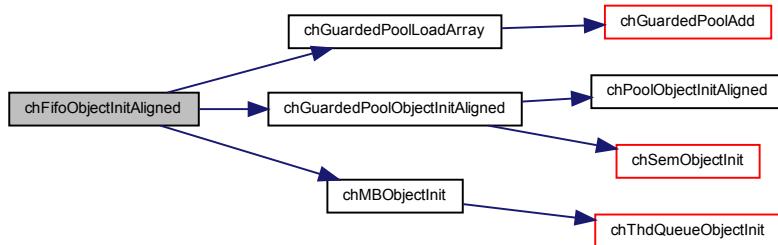
Parameters

out	<i>ofp</i>	pointer to a objects_fifo_t structure
in	<i>objsize</i>	size of objects
in	<i>objn</i>	number of objects available
in	<i>objalign</i>	required objects alignment
in	<i>objbuf</i>	pointer to the buffer of objects, it must be able to hold <i>objn</i> objects of <i>objszie</i> size with <i>objalign</i> alignment
in	<i>msgbuf</i>	pointer to the buffer of messages, it must be able to hold <i>objn</i> messages

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:

**7.50.3.2 chFifoObjectInit()**

```

static void chFifoObjectInit (
    objects_fifo_t * ofp,
    size_t objsize,
    size_t objn,
    void * objbuf,
    msg_t * msgbuf ) [inline], [static]
  
```

Initializes a FIFO object.

Precondition

The messages size must be a multiple of the alignment requirement.

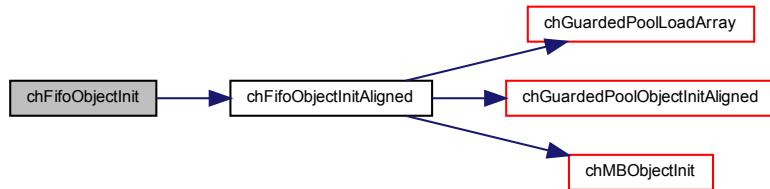
Parameters

out	<code>ofp</code>	pointer to a <code>objects_fifo_t</code> structure
in	<code>objsize</code>	size of objects
in	<code>objn</code>	number of objects available
in	<code>objbuf</code>	pointer to the buffer of objects, it must be able to hold <code>objn</code> objects of <code>objsize</code> size
in	<code>msgbuf</code>	pointer to the buffer of messages, it must be able to hold <code>objn</code> messages

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:

**7.50.3.3 chFifoTakeObjectI()**

```
static void* chFifoTakeObjectI (
    objects_fifo_t * ofp ) [inline], [static]
```

Allocates a free object.

Parameters

in	<i>ofp</i>	pointer to a objects_fifo_t structure
----	------------	---------------------------------------

Returns

The pointer to the allocated object.

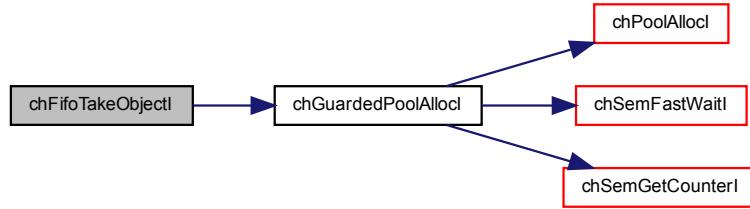
Return values

NULL	if an object is not immediately available.
------	--

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:

**7.50.3.4 chFifoTakeObjectTimeoutS()**

```

static void* chFifoTakeObjectTimeoutS (
    objects_fifo_t * ofp,
    sysinterval_t timeout ) [inline], [static]
  
```

Allocates a free object.

Parameters

in	<i>ofp</i>	pointer to a <code>objects_fifo_t</code> structure
in	<i>timeout</i>	the number of ticks before the operation timeouts, the following special values are allowed: <ul style="list-style-type: none"> • <code>TIME_IMMEDIATE</code> immediate timeout. • <code>TIME_INFINITE</code> no timeout.

Returns

The pointer to the allocated object.

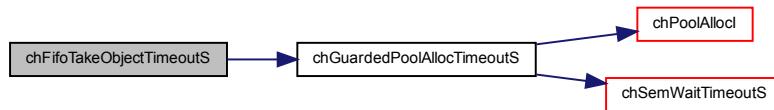
Return values

<code>NULL</code>	if an object is not available 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:



7.50.3.5 chFifoTakeObjectTimeout()

```
static void* chFifoTakeObjectTimeout (
    objects_fifo_t * ofp,
    sysinterval_t timeout ) [inline], [static]
```

Allocates a free object.

Parameters

in	<i>ofp</i>	pointer to a <code>objects_fifo_t</code> structure
in	<i>timeout</i>	the number of ticks before the operation timeouts, the following special values are allowed: <ul style="list-style-type: none"> • <code>TIME_IMMEDIATE</code> immediate timeout. • <code>TIME_INFINITE</code> no timeout.

Returns

The pointer to the allocated object.

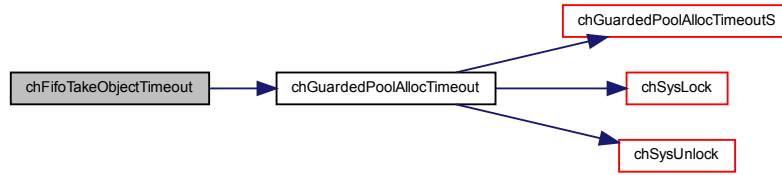
Return values

<code>NULL</code>	if an object is not available 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:

**7.50.3.6 chFifoReturnObjectI()**

```
static void chFifoReturnObjectI (
    objects_fifo_t * ofp,
    void * objp ) [inline], [static]
```

Releases a fetched object.

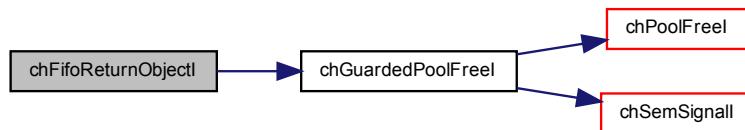
Parameters

in	<i>ofp</i>	pointer to a <code>objects_fifo_t</code> structure
in	<i>objp</i>	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:



7.50.3.7 chFifoReturnObjectS()

```
static void chFifoReturnObjects (
    objects_fifo_t * ofp,
    void * objp ) [inline], [static]
```

Releases a fetched object.

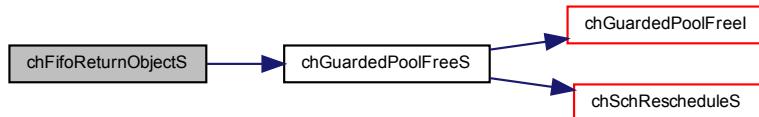
Parameters

in	<i>ofp</i>	pointer to a objects_fifo_t structure
in	<i>objp</i>	pointer to the object to be released

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:



7.50.3.8 chFifoReturnObject()

```
static void chFifoReturnObject (
    objects_fifo_t * ofp,
    void * objp ) [inline], [static]
```

Releases a fetched object.

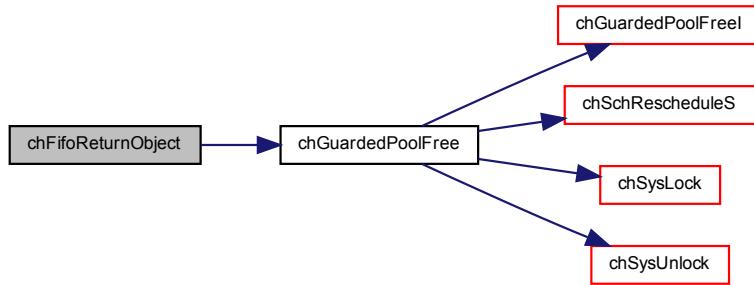
Parameters

in	<i>ofp</i>	pointer to a objects_fifo_t structure
in	<i>objp</i>	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:

**7.50.3.9 chFifoSendObjectI()**

```

static void chFifoSendObjectI (
    objects_fifo_t * ofp,
    void * objp ) [inline], [static]
  
```

Posts an object.

Note

By design the object can be always immediately posted.

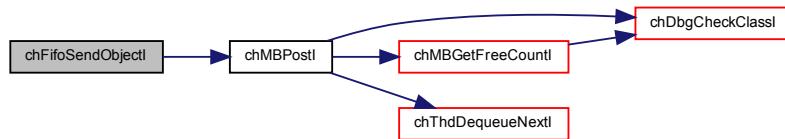
Parameters

in	<i>ofp</i>	pointer to a <code>objects_fifo_t</code> structure
in	<i>objp</i>	pointer to the object to 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:



7.50.3.10 chFifoSendObjectS()

```
static void chFifoSendObjectS (
    objects_fifo_t * ofp,
    void * objp ) [inline], [static]
```

Posts an object.

Note

By design the object can be always immediately posted.

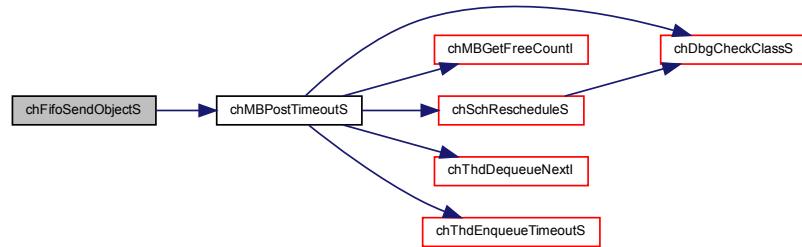
Parameters

in	<i>ofp</i>	pointer to a <code>objects_fifo_t</code> structure
in	<i>objp</i>	pointer to the object to be posted

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:

**7.50.3.11 chFifoSendObject()**

```

static void chFifoSendObject (
    objects_fifo_t * ofp,
    void * objp ) [inline], [static]
  
```

Posts an object.

Note

By design the object can be always immediately posted.

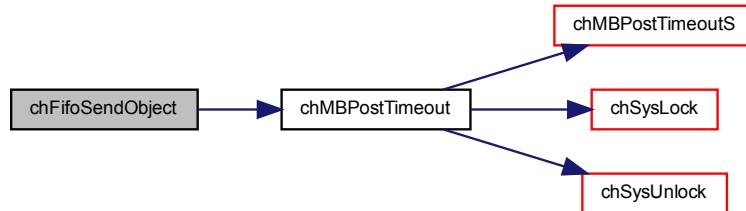
Parameters

in	<i>ofp</i>	pointer to a <code>objects_fifo_t</code> structure
in	<i>objp</i>	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:

**7.50.3.12 chFifoSendObjectAheadI()**

```
static void chFifoSendObjectAheadI (
    objects_fifo_t * ofp,
    void * objp ) [inline], [static]
```

Posts an high priority object.

Note

By design the object can be always immediately posted.

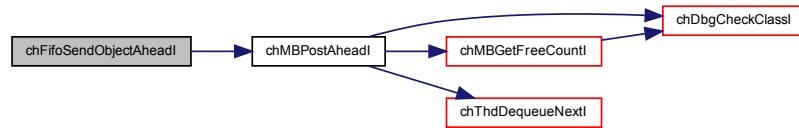
Parameters

in	<i>ofp</i>	pointer to a <code>objects_fifo_t</code> structure
in	<i>objp</i>	pointer to the object to 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:

**7.50.3.13 chFifoSendObjectAheadS()**

```

static void chFifoSendObjectAheadS (
    objects_fifo_t * ofp,
    void * objp ) [inline], [static]

```

Posts an high priority object.

Note

By design the object can be always immediately posted.

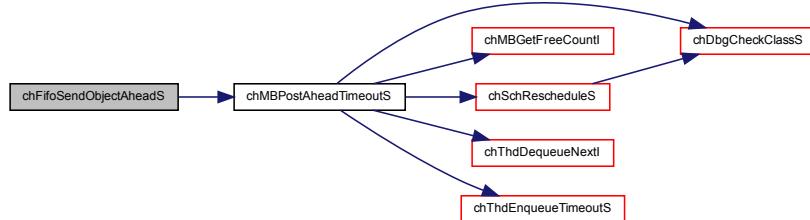
Parameters

in	<i>ofp</i>	pointer to a <code>objects_fifo_t</code> structure
in	<i>objp</i>	pointer to the object to be posted

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:



7.50.3.14 chFifoSendObjectAhead()

```
static void chFifoSendObjectAhead (
    objects_fifo_t * ofp,
    void * objp ) [inline], [static]
```

Posts an high priority object.

Note

By design the object can be always immediately posted.

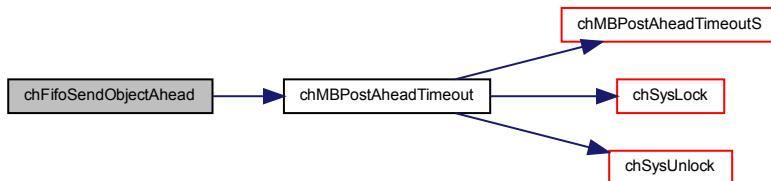
Parameters

in	<i>ofp</i>	pointer to a <code>objects_fifo_t</code> structure
in	<i>objp</i>	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:



7.50.3.15 chFifoReceiveObjectI()

```
static msg_t chFifoReceiveObjectI (
    objects_fifo_t * ofp,
    void ** objpp ) [inline], [static]
```

Fetches an object.

Parameters

in	<i>ofp</i>	pointer to a <code>objects_fifo_t</code> structure
in	<i>objpp</i>	pointer to the fetched object reference

Returns

The operation status.

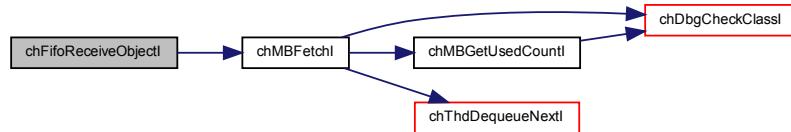
Return values

<i>MSG_OK</i>	if an object has been correctly fetched.
<i>MSG_TIMEOUT</i>	if the FIFO 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:

**7.50.3.16 chFifoReceiveObjectTimeoutS()**

```

static msg_t chFifoReceiveObjectTimeouts (
    objects_fifo_t * ofp,
    void ** objpp,
    sysinterval_t timeout ) [inline], [static]
  
```

Fetches an object.

Parameters

in	<i>ofp</i>	pointer to a <code>objects_fifo_t</code> structure
in	<i>objpp</i>	pointer to the fetched object reference
in	<i>timeout</i>	the number of ticks before the operation timeouts, the following special values are allowed: <ul style="list-style-type: none"> • <code>TIME_IMMEDIATE</code> immediate timeout. • <code>TIME_INFINITE</code> no timeout.

Returns

The operation status.

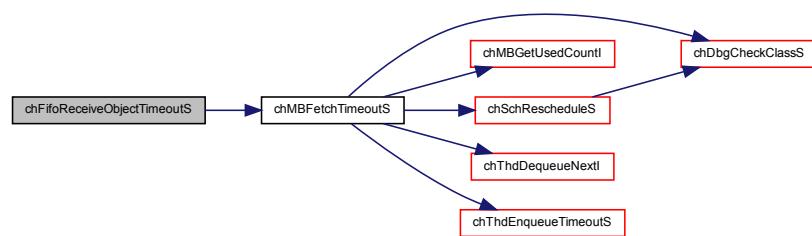
Return values

<code>MSG_OK</code>	if an object has been correctly fetched.
<code>MSG_TIMEOUT</code>	if the operation has timed out.

Function Class:

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

Here is the call graph for this function:



7.50.3.17 chFifoReceiveObjectTimeout()

```
static msg_t chFifoReceiveObjectTimeout (
    objects_fifo_t * ofp,
    void ** objpp,
    sysinterval_t timeout ) [inline], [static]
```

Fetches an object.

Parameters

in	<code>ofp</code>	pointer to a <code>objects_fifo_t</code> structure
in	<code>objpp</code>	pointer to the fetched object reference
in	<code>timeout</code>	the number of ticks before the operation timeouts, the following special values are allowed: <ul style="list-style-type: none"> • <code>TIME_IMMEDIATE</code> immediate timeout. • <code>TIME_INFINITE</code> no timeout.

Returns

The operation status.

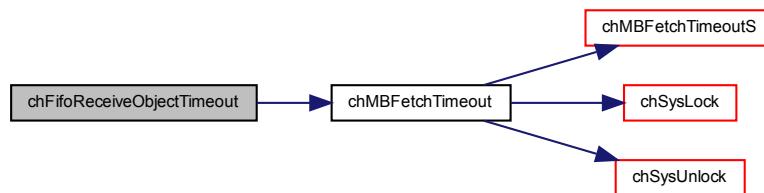
Return values

<i>MSG_OK</i>	if an object has been correctly fetched.
<i>MSG_TIMEOUT</i>	if the operation has timed out.

Function Class:

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

Here is the call graph for this function:



7.51 Objects Caches

7.51.1 Detailed Description

Cached objects flags

- #define **OC_FLAG_INLRU** 0x00000001U
- #define **OC_FLAG_INHASH** 0x00000002U
- #define **OC_FLAG_SHARED** 0x00000004U
- #define **OC_FLAG_NOTSYNC** 0x00000008U
- #define **OC_FLAG_LAZYWRITE** 0x00000010U
- #define **OC_FLAG_FORGET** 0x00000020U

Typedefs

- typedef uint32_t **oc_flags_t**
Flags of cached objects.
- typedef struct **ch_oc_hash_header** **oc_hash_header_t**
Type of an hash element header.
- typedef struct **ch_oc_lru_header** **oc_lru_header_t**
Type of an LRU element header.
- typedef struct **ch_oc_object** **oc_object_t**
Type of a cached object.
- typedef struct **ch_objects_cache** **objects_cache_t**
Type of a cache object.
- typedef bool(* **oc_readf_t**) (**objects_cache_t** *ocp, **oc_object_t** *objp, bool async)
Object read function.
- typedef bool(* **oc_writef_t**) (**objects_cache_t** *ocp, **oc_object_t** *objp, bool async)
Object write function.

Data Structures

- struct **ch_oc_hash_header**
Structure representing an hash table element.
- struct **ch_oc_lru_header**
Structure representing an hash table element.
- struct **ch_oc_object**
Structure representing a cached object.
- struct **ch_objects_cache**
Structure representing a cache object.

Functions

- static `oc_object_t * hash_get_s (objects_cache_t *ocp, uint32_t group, uint32_t key)`
Returns an object pointer from the cache, if present.
- static `oc_object_t * lru_get_last_s (objects_cache_t *ocp)`
Gets the least recently used object buffer from the LRU list.
- void `chCacheObjectInit (objects_cache_t *ocp, ucnt_t hashn, oc_hash_header_t *hashp, ucnt_t objn, size_t objsz, void *objvp, oc_readf_t readf, oc_writef_t writef)`
Initializes a objects_cache_t object.
- `oc_object_t * chCacheGetObject (objects_cache_t *ocp, uint32_t group, uint32_t key)`
Retrieves an object from the cache.
- void `chCacheReleaseObject (objects_cache_t *ocp, oc_object_t *objp)`
Releases an object into the cache.
- bool `chCacheReadObject (objects_cache_t *ocp, oc_object_t *objp, bool async)`
Reads object data from the storage.
- bool `chCacheWriteObject (objects_cache_t *ocp, oc_object_t *objp, bool async)`
Writes the object data back to storage.
- static void `chCacheReleaseObject (objects_cache_t *ocp, oc_object_t *objp)`
Releases an object into the cache.

7.51.2 Typedef Documentation

7.51.2.1 `oc_flags_t`

```
typedef uint32_t oc_flags_t
```

Flags of cached objects.

7.51.2.2 `oc_hash_header_t`

```
typedef struct ch_oc_hash_header oc_hash_header_t
```

Type of an hash element header.

7.51.2.3 `oc_lru_header_t`

```
typedef struct ch_oc_lru_header oc_lru_header_t
```

Type of an LRU element header.

7.51.2.4 `oc_object_t`

```
typedef struct ch_oc_object oc_object_t
```

Type of a cached object.

7.51.2.5 `objects_cache_t`

```
typedef struct ch_objects_cache objects_cache_t
```

Type of a cache object.

7.51.2.6 `oc_readf_t`

```
typedef bool(* oc_readf_t) (objects_cache_t *ocp, oc_object_t *objp, bool async)
```

Object read function.

Parameters

in	<i>ocp</i>	pointer to the <code>objects_cache_t</code> structure
in	<i>async</i>	requests an asynchronous operation if supported, the function is then responsible for releasing the object

7.51.2.7 `oc_writef_t`

```
typedef bool(* oc_writef_t) (objects_cache_t *ocp, oc_object_t *objp, bool async)
```

Object write function.

Parameters

in	<i>ocp</i>	pointer to the <code>objects_cache_t</code> structure
in	<i>async</i>	requests an asynchronous operation if supported, the function is then responsible for releasing the object

7.51.3 Function Documentation

7.51.3.1 hash_get_s()

```
static oc_object_t* hash_get_s (
    objects_cache_t * ocp,
    uint32_t group,
    uint32_t key ) [static]
```

Returns an object pointer from the cache, if present.

Parameters

out	<i>ocp</i>	pointer to the <code>objects_cache_t</code> structure to be
in	<i>group</i>	object group identifier
in	<i>key</i>	object identifier within the group initialized

Returns

The pointer to the retrieved object.

Return values

<code>NULL</code>	if the object is not in cache.
-------------------	--------------------------------

Function Class:

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

7.51.3.2 lru_get_last_s()

```
static oc_object_t* lru_get_last_s (
    objects_cache_t * ocp ) [static]
```

Gets the least recently used object buffer from the LRU list.

Parameters

out	<i>ocp</i>	pointer to the <code>objects_cache_t</code> structure to be
-----	------------	---

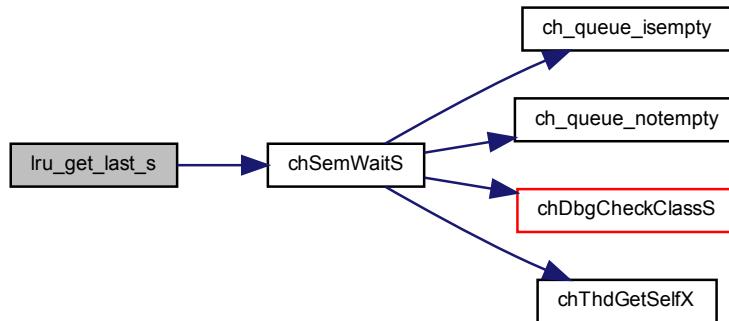
Returns

The pointer to the retrieved object.

Function Class:

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

Here is the call graph for this function:



7.51.3.3 chCacheObjectInit()

```

void chCacheObjectInit (
    objects_cache_t * ocp,
    ucnt_t hashn,
    oc_hash_header_t * hashp,
    ucnt_t objn,
    size_t objsz,
    void * objvp,
    oc_readf_t readf,
    oc_writef_t writef )
  
```

Initializes a `objects_cache_t` object.

Parameters

<code>out</code>	<code>ocp</code>	pointer to the <code>objects_cache_t</code> structure to be initialized
<code>in</code>	<code>hashn</code>	number of elements in the hash table array, must be a power of two and not lower than <code>objn</code>
<code>in</code>	<code>hashp</code>	pointer to the hash table as an array of <code>oc_hash_header_t</code>
<code>in</code>	<code>objn</code>	number of elements in the objects table array
<code>in</code>	<code>objsz</code>	size of elements in the objects table array, the minimum value is <code>sizeof (oc_object_t)</code> .
<code>in</code>	<code>objvp</code>	pointer to the hash objects as an array of structures starting with an <code>oc_object_t</code>
<code>in</code>	<code>readf</code>	pointer to an object reader function
<code>in</code>	<code>writef</code>	pointer to an object writer function

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:

**7.51.3.4 chCacheGetObject()**

```
oc_object_t * chCacheGetObject (
    objects_cache_t * ocp,
    uint32_t group,
    uint32_t key )
```

Retrieves an object from the cache.

Note

If the object is not in cache then the returned object is marked as `OC_FLAG_NOTSYNC` meaning that its data contains garbage and must be initialized.

Parameters

in	<i>ocp</i>	pointer to the <code>objects_cache_t</code> structure
in	<i>group</i>	object group identifier
in	<i>key</i>	object identifier within the group

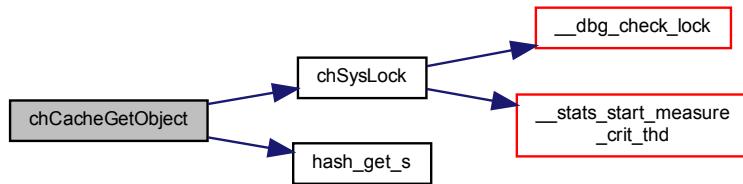
Returns

The pointer to the retrieved 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:

**7.51.3.5 chCacheReleaseObjectI()**

```
void chCacheReleaseObjectI (
    objects_cache_t * ocp,
    oc_object_t * objp )
```

Releases an object into the cache.

Note

This function gives a meaning to the following flags:

- `OC_FLAG_INLRU` must be cleared.
- `OC_FLAG_INHASH` must be set.
- `OC_FLAG_SHARED` must be cleared.
- `OC_FLAG_NOTSYNC` invalidates the object and queues it on the LRU tail.
- `OC_FLAG_LAZYWRITE` is ignored and kept, a write will occur when the object is removed from the LRU list (lazy write).

Parameters

in	<code>ocp</code>	pointer to the <code>objects_cache_t</code> structure
in	<code>objp</code>	pointer to the <code>oc_object_t</code> structure

Function Class:

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

7.51.3.6 chCacheReadObject()

```
bool chCacheReadObject (
    objects_cache_t * ocp,
    oc_object_t * objp,
    bool async )
```

Reads object data from the storage.

Note

In case of asynchronous operation an error condition is not reported by this function.

Parameters

in	<i>ocp</i>	pointer to the <code>objects_cache_t</code> structure
in	<i>objp</i>	pointer to the <code>oc_object_t</code> structure
in	<i>async</i>	requests an asynchronous operation if supported, the function is then responsible for releasing the object

Returns

The operation status. In case of asynchronous operation `false` is always returned.

Return values

<code>false</code>	if the operation succeeded.
<code>true</code>	if the synchronous read operation failed.

Function Class:

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

7.51.3.7 chCacheWriteObject()

```
bool chCacheWriteObject (
    objects_cache_t * ocp,
    oc_object_t * objp,
    bool async )
```

Writes the object data back to storage.

Note

In case of asynchronous operation an error condition is not reported by this function.

Parameters

in	<i>ocp</i>	pointer to the <code>objects_cache_t</code> structure
in	<i>objp</i>	pointer to the <code>oc_object_t</code> structure
in	<i>async</i>	requests an asynchronous operation if supported, the function is then responsible for releasing the object

Returns

The operation status. In case of asynchronous operation `false` is always returned.

Return values

<i>false</i>	if the operation succeeded.
<i>true</i>	if the synchronous write operation failed.

Function Class:

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

7.51.3.8 chCacheReleaseObject()

```
static void chCacheReleaseObject (
    objects_cache_t * ocp,
    oc_object_t * objp ) [inline], [static]
```

Releases an object into the cache.

Note

This function gives a meaning to the following flags:

- `OC_FLAG_INLRU` must be cleared.
- `OC_FLAG_INHASH` must be set.
- `OC_FLAG_SHARED` must be cleared.
- `OC_FLAG_NOTSYNC` invalidates the object and queues it on the LRU tail.
- `OC_FLAG_LAZYWRITE` is ignored and kept, a write will occur when the object is removed from the LRU list (lazy write).

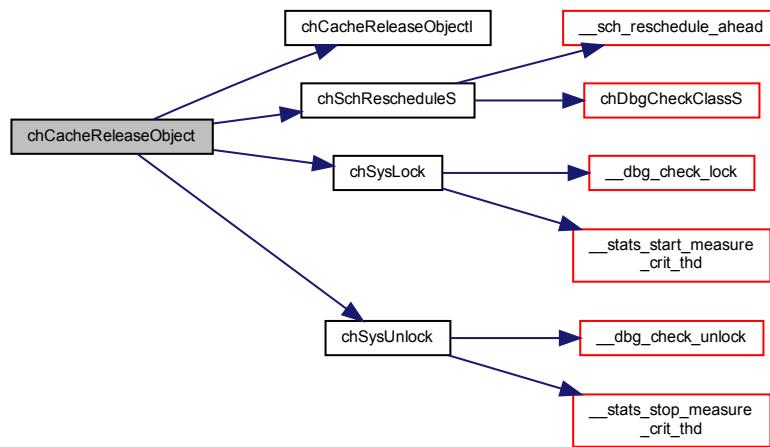
Parameters

in	<i>ocp</i>	pointer to the <code>objects_cache_t</code> structure
in	<i>objp</i>	pointer to the <code>oc_object_t</code> structure

Function Class:

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

Here is the call graph for this function:



7.52 Dynamic Objects Factory

7.52.1 Detailed Description

The object factory is a subsystem that allows to:

- Register static objects by name.
- Dynamically create objects and assign them a name.
- Retrieve existing objects by name.
- Free objects by reference.

Allocated OS objects are handled using a reference counter, only when all references have been released then the object memory is freed in a pool.

Precondition

This subsystem requires the `CH_CFG_USE_MEMCORE` and `CH_CFG_USE_MEMPOOLS` options to be set to TRUE. The option `CH_CFG_USE_HEAP` is also required if the support for variable length objects is enabled.

Note

Compatible with RT and NIL.

Macros

- `#define CH_CFG_FACTORY_MAX_NAMES_LENGTH 8`
Maximum length for object names.
- `#define CH_CFG_FACTORY_OBJECTS_REGISTRY TRUE`
Enables the registry of generic objects.
- `#define CH_CFG_FACTORY_GENERIC_BUFFERS TRUE`
Enables factory for generic buffers.
- `#define CH_CFG_FACTORY_SEMAPHORES TRUE`
Enables factory for semaphores.
- `#define CH_CFG_FACTORY_SEMAPHORES FALSE`
Enables factory for semaphores.
- `#define CH_CFG_FACTORY_MAILBOXES TRUE`
Enables factory for mailboxes.
- `#define CH_CFG_FACTORY_MAILBOXES FALSE`
Enables factory for mailboxes.
- `#define CH_CFG_FACTORY_OBJ_FIFOS TRUE`
Enables factory for objects FIFOs.
- `#define CH_CFG_FACTORY_OBJ_FIFOS TRUE`
Enables factory for objects FIFOs.
- `#define CH_CFG_FACTORY_OBJ_FIFOS FALSE`
Enables factory for objects FIFOs.
- `#define CH_CFG_FACTORY_PIPES TRUE`
Enables factory for Pipes.
- `#define CH_CFG_FACTORY_PIPES FALSE`
Enables factory for Pipes.

Typedefs

- **typedef struct ch_dyn_element dyn_element_t**
Type of a dynamic object list element.
- **typedef struct ch_dyn_list dyn_list_t**
Type of a dynamic object list.
- **typedef struct ch_registered_static_object registered_object_t**
Type of a registered object.
- **typedef struct ch_dyn_object dyn_buffer_t**
Type of a dynamic buffer object.
- **typedef struct ch_dyn_semaphore dyn_semaphore_t**
Type of a dynamic semaphore.
- **typedef struct ch_dyn_mailbox dyn_mailbox_t**
Type of a dynamic buffer object.
- **typedef struct ch_dyn_objects_fifo dyn_objects_fifo_t**
Type of a dynamic buffer object.
- **typedef struct ch_dyn_pipe dyn_pipe_t**
Type of a dynamic pipe object.
- **typedef struct ch_objects_factory objects_factory_t**
Type of the factory main object.

Data Structures

- **struct ch_dyn_element**
Type of a dynamic object list element.
- **struct ch_dyn_list**
Type of a dynamic object list.
- **struct ch_registered_static_object**
Type of a registered object.
- **struct ch_dyn_object**
Type of a dynamic buffer object.
- **struct ch_dyn_semaphore**
Type of a dynamic semaphore.
- **struct ch_dyn_mailbox**
Type of a dynamic buffer object.
- **struct ch_dyn_objects_fifo**
Type of a dynamic buffer object.
- **struct ch_dyn_pipe**
Type of a dynamic pipe object.
- **struct ch_objects_factory**
Type of the factory main object.

Functions

- void `__factory_init` (void)

Initializes the objects factory.
- `registered_object_t * chFactoryRegisterObject` (const char *name, void *objp)

Registers a generic object.
- `registered_object_t * chFactoryFindObject` (const char *name)

Retrieves a registered object.
- `registered_object_t * chFactoryFindObjectByPointer` (void *objp)

Retrieves a registered object by pointer.
- void `chFactoryReleaseObject` (`registered_object_t` *rop)

Releases a registered object.
- `dyn_buffer_t * chFactoryCreateBuffer` (const char *name, size_t size)

Creates a generic dynamic buffer object.
- `dyn_buffer_t * chFactoryFindBuffer` (const char *name)

Retrieves a dynamic buffer object.
- void `chFactoryReleaseBuffer` (`dyn_buffer_t` *dbp)

Releases a dynamic buffer object.
- `dyn_semaphore_t * chFactoryCreateSemaphore` (const char *name, cnt_t n)

Creates a dynamic semaphore object.
- `dyn_semaphore_t * chFactoryFindSemaphore` (const char *name)

Retrieves a dynamic semaphore object.
- void `chFactoryReleaseSemaphore` (`dyn_semaphore_t` *dsp)

Releases a dynamic semaphore object.
- `dyn_mailbox_t * chFactoryCreateMailbox` (const char *name, size_t n)

Creates a dynamic mailbox object.
- `dyn_mailbox_t * chFactoryFindMailbox` (const char *name)

Retrieves a dynamic mailbox object.
- void `chFactoryReleaseMailbox` (`dyn_mailbox_t` *dmp)

Releases a dynamic mailbox object.
- `dyn_objects_fifo_t * chFactoryCreateObjectsFIFO` (const char *name, size_t objsize, size_t objn, unsigned objalign)

Creates a dynamic "objects FIFO" object.
- `dyn_objects_fifo_t * chFactoryFindObjectObjectsFIFO` (const char *name)

Retrieves a dynamic "objects FIFO" object.
- void `chFactoryReleaseObjectsFIFO` (`dyn_objects_fifo_t` *dofp)

Releases a dynamic "objects FIFO" object.
- `dyn_pipe_t * chFactoryCreatePipe` (const char *name, size_t size)

Creates a dynamic pipe object.
- `dyn_pipe_t * chFactoryFindPipe` (const char *name)

Retrieves a dynamic pipe object.
- void `chFactoryReleasePipe` (`dyn_pipe_t` *dpp)

Releases a dynamic pipe object.
- static `dyn_element_t * chFactoryDuplicateReference` (`dyn_element_t` *dep)

Duplicates an object reference.
- static void * `chFactoryGetObject` (`registered_object_t` *rop)

Returns the pointer to the inner registered object.
- static size_t `chFactoryGetBufferSize` (`dyn_buffer_t` *dbp)

Returns the size of a generic dynamic buffer object.
- static uint8_t * `chFactoryGetBuffer` (`dyn_buffer_t` *dbp)

Returns the pointer to the inner buffer.

- static `semaphore_t * chFactoryGetSemaphore (dyn_semaphore_t *dsp)`
Returns the pointer to the inner semaphore.
- static `mailbox_t * chFactoryGetMailbox (dyn_mailbox_t *dmp)`
Returns the pointer to the inner mailbox.
- static `objects_fifo_t * chFactoryGetObjectsFIFO (dyn_objects_fifo_t *dofp)`
Returns the pointer to the inner objects FIFO.
- static `pipe_t * chFactoryGetPipe (dyn_pipe_t *dpp)`
Returns the pointer to the inner pipe.

Variables

- `objects_factory_t ch_factory`
Factory object static instance.

7.52.2 Macro Definition Documentation

7.52.2.1 CH_CFG_FACTORY_MAX_NAMES_LENGTH

```
#define CH_CFG_FACTORY_MAX_NAMES_LENGTH 8
```

Maximum length for object names.

If the specified length is zero then the name is stored by pointer but this could have unintended side effects.

7.52.2.2 CH_CFG_FACTORY_OBJECTS_REGISTRY

```
#define CH_CFG_FACTORY_OBJECTS_REGISTRY TRUE
```

Enables the registry of generic objects.

7.52.2.3 CH_CFG_FACTORY_GENERIC_BUFFERS

```
#define CH_CFG_FACTORY_GENERIC_BUFFERS TRUE
```

Enables factory for generic buffers.

7.52.2.4 CH_CFG_FACTORY_SEMAPHORES [1/2]

```
#define CH_CFG_FACTORY_SEMAPHORES TRUE
```

Enables factory for semaphores.

7.52.2.5 CH_CFG_FACTORY_SEMAPHORES [2/2]

```
#define CH_CFG_FACTORY_SEMAPHORES FALSE
```

Enables factory for semaphores.

7.52.2.6 CH_CFG_FACTORY_MAILBOXES [1/2]

```
#define CH_CFG_FACTORY_MAILBOXES TRUE
```

Enables factory for mailboxes.

7.52.2.7 CH_CFG_FACTORY_MAILBOXES [2/2]

```
#define CH_CFG_FACTORY_MAILBOXES FALSE
```

Enables factory for mailboxes.

7.52.2.8 CH_CFG_FACTORY_OBJ_FIFOS [1/3]

```
#define CH_CFG_FACTORY_OBJ_FIFOS TRUE
```

Enables factory for objects FIFOs.

7.52.2.9 CH_CFG_FACTORY_OBJ_FIFOS [2/3]

```
#define CH_CFG_FACTORY_OBJ_FIFOS TRUE
```

Enables factory for objects FIFOs.

7.52.2.10 CH_CFG_FACTORY_OBJ_FIFOS [3/3]

```
#define CH_CFG_FACTORY_OBJ_FIFOS FALSE
```

Enables factory for objects FIFOs.

7.52.2.11 CH_CFG_FACTORY_PIPES [1/2]

```
#define CH_CFG_FACTORY_PIPES TRUE
```

Enables factory for Pipes.

7.52.2.12 CH_CFG_FACTORY_PIPES [2/2]

```
#define CH_CFG_FACTORY_PIPES FALSE
```

Enables factory for Pipes.

7.52.3 Typedef Documentation**7.52.3.1 dyn_element_t**

```
typedef struct ch_dyn_element dyn_element_t
```

Type of a dynamic object list element.

7.52.3.2 dyn_list_t

```
typedef struct ch_dyn_list dyn_list_t
```

Type of a dynamic object list.

7.52.3.3 registered_object_t

```
typedef struct ch_registered_static_object registered_object_t
```

Type of a registered object.

7.52.3.4 dyn_buffer_t

```
typedef struct ch_dyn_object dyn_buffer_t
```

Type of a dynamic buffer object.

7.52.3.5 `dyn_semaphore_t`

```
typedef struct ch_dyn_semaphore dyn_semaphore_t
```

Type of a dynamic semaphore.

7.52.3.6 `dyn_mailbox_t`

```
typedef struct ch_dyn_mailbox dyn_mailbox_t
```

Type of a dynamic buffer object.

7.52.3.7 `dyn_objects_fifo_t`

```
typedef struct ch_dyn_objects_fifo dyn_objects_fifo_t
```

Type of a dynamic buffer object.

7.52.3.8 `dyn_pipe_t`

```
typedef struct ch_dyn_pipe dyn_pipe_t
```

Type of a dynamic pipe object.

7.52.3.9 `objects_factory_t`

```
typedef struct ch_objects_factory objects_factory_t
```

Type of the factory main object.

7.52.4 Function Documentation

7.52.4.1 __factory_init()

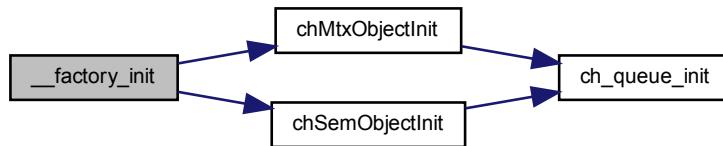
```
void __factory_init (
    void )
```

Initializes the objects factory.

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:



7.52.4.2 chFactoryRegisterObject()

```
registered_object_t * chFactoryRegisterObject (
    const char * name,
    void * objp )
```

Registers a generic object.

Postcondition

A reference to the registered object is returned and the reference counter is initialized to one.

Parameters

in	<i>name</i>	name to be assigned to the registered object
in	<i>objp</i>	pointer to the object to be registered

Returns

The reference to the registered object.

Return values

<code>NULL</code>	if the object to be registered cannot be allocated or a registered object with the same name exists.
-------------------	--

Function Class:

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

7.52.4.3 chFactoryFindObject()

```
registered_object_t * chFactoryFindObject (
    const char * name )
```

Retrieves a registered object.

Postcondition

A reference to the registered object is returned with the reference counter increased by one.

Parameters

<code>in</code>	<code>name</code>	name of the registered object
-----------------	-------------------	-------------------------------

Returns

The reference to the found registered object.

Return values

<code>NULL</code>	if a registered object with the specified name does not exist.
-------------------	--

Function Class:

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

7.52.4.4 chFactoryFindObjectByPointer()

```
registered_object_t * chFactoryFindObjectByPointer (
    void * objp )
```

Retrieves a registered object by pointer.

Postcondition

A reference to the registered object is returned with the reference counter increased by one.

Parameters

in	<i>objp</i>	pointer to the object to be retrieved
----	-------------	---------------------------------------

Returns

The reference to the found registered object.

Return values

NULL	if a registered object with the specified pointer does not exist.
------	---

Function Class:

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

7.52.4.5 chFactoryReleaseObject()

```
void chFactoryReleaseObject (
    registered_object_t * rop )
```

Releases a registered object.

The reference counter of the registered object is decreased by one, if reaches zero then the registered object memory is freed.

Note

The object itself is not freed, it could be static, only the allocated list element is freed.

Parameters

in	<i>rop</i>	registered object reference
----	------------	-----------------------------

Function Class:

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

7.52.4.6 chFactoryCreateBuffer()

```
dyn_buffer_t * chFactoryCreateBuffer (
    const char * name,
    size_t size )
```

Creates a generic dynamic buffer object.

Postcondition

A reference to the dynamic buffer object is returned and the reference counter is initialized to one.
 The dynamic buffer object is filled with zeros.

Parameters

in	<i>name</i>	name to be assigned to the new dynamic buffer object
in	<i>size</i>	payload size of the dynamic buffer object to be created

Returns

The reference to the created dynamic buffer object.

Return values

<i>NULL</i>	if the dynamic buffer object cannot be allocated or a dynamic buffer object with the same name exists.
-------------	--

Function Class:

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

7.52.4.7 chFactoryFindBuffer()

```
dyn_buffer_t * chFactoryFindBuffer (
    const char * name )
```

Retrieves a dynamic buffer object.

Postcondition

A reference to the dynamic buffer object is returned with the reference counter increased by one.

Parameters

in	<i>name</i>	name of the dynamic buffer object
----	-------------	-----------------------------------

Returns

The reference to the found dynamic buffer object.

Return values

<i>NULL</i>	if a dynamic buffer object with the specified name does not exist.
-------------	--

Function Class:

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

7.52.4.8 chFactoryReleaseBuffer()

```
void chFactoryReleaseBuffer (
    dyn_buffer_t * dbp )
```

Releases a dynamic buffer object.

The reference counter of the dynamic buffer object is decreased by one, if reaches zero then the dynamic buffer object memory is freed.

Parameters

in	<i>dbp</i>	dynamic buffer object reference
----	------------	---------------------------------

Function Class:

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

7.52.4.9 chFactoryCreateSemaphore()

```
dyn_semaphore_t * chFactoryCreateSemaphore (
    const char * name,
    cnt_t n )
```

Creates a dynamic semaphore object.

Postcondition

A reference to the dynamic semaphore object is returned and the reference counter is initialized to one.

The dynamic semaphore object is initialized and ready to use.

Parameters

in	<i>name</i>	name to be assigned to the new dynamic semaphore object
in	<i>n</i>	dynamic semaphore object counter initialization value

Returns

The reference to the created dynamic semaphore object.

Return values

<code>NULL</code>	if the dynamic semaphore object cannot be allocated or a dynamic semaphore with the same name exists.
-------------------	---

Function Class:

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

7.52.4.10 chFactoryFindSemaphore()

```
dyn_semaphore_t * chFactoryFindSemaphore (
    const char * name )
```

Retrieves a dynamic semaphore object.

Postcondition

A reference to the dynamic semaphore object is returned with the reference counter increased by one.

Parameters

<code>in</code>	<code>name</code>	name of the dynamic semaphore object
-----------------	-------------------	--------------------------------------

Returns

The reference to the found dynamic semaphore object.

Return values

<code>NULL</code>	if a dynamic semaphore object with the specified name does not exist.
-------------------	---

Function Class:

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

7.52.4.11 chFactoryReleaseSemaphore()

```
void chFactoryReleaseSemaphore (
    dyn_semaphore_t * dsp )
```

Releases a dynamic semaphore object.

The reference counter of the dynamic semaphore object is decreased by one, if reaches zero then the dynamic semaphore object memory is freed.

Parameters

in	<i>dsp</i>	dynamic semaphore object reference
----	------------	------------------------------------

Function Class:

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

7.52.4.12 chFactoryCreateMailbox()

```
dyn_mailbox_t * chFactoryCreateMailbox (
    const char * name,
    size_t n )
```

Creates a dynamic mailbox object.

Postcondition

A reference to the dynamic mailbox object is returned and the reference counter is initialized to one.

The dynamic mailbox object is initialized and ready to use.

Parameters

in	<i>name</i>	name to be assigned to the new dynamic mailbox object
in	<i>n</i>	mailbox buffer size as number of messages

Returns

The reference to the created dynamic mailbox object.

Return values

NULL	if the dynamic mailbox object cannot be allocated or a dynamic mailbox object with the same name exists.
------	--

Function Class:

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

7.52.4.13 chFactoryFindMailbox()

```
dyn_mailbox_t * chFactoryFindMailbox (
    const char * name )
```

Retrieves a dynamic mailbox object.

Postcondition

A reference to the dynamic mailbox object is returned with the reference counter increased by one.

Parameters

in	<i>name</i>	name of the dynamic mailbox object
----	-------------	------------------------------------

Returns

The reference to the found dynamic mailbox object.

Return values

<i>NULL</i>	if a dynamic mailbox object with the specified name does not exist.
-------------	---

Function Class:

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

7.52.4.14 chFactoryReleaseMailbox()

```
void chFactoryReleaseMailbox (
    dyn_mailbox_t * dmp )
```

Releases a dynamic mailbox object.

The reference counter of the dynamic mailbox object is decreased by one, if reaches zero then the dynamic mailbox object memory is freed.

Parameters

in	<i>dmp</i>	dynamic mailbox object reference
----	------------	----------------------------------

Function Class:

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

7.52.4.15 chFactoryCreateObjectsFIFO()

```
dyn_objects_fifo_t * chFactoryCreateObjectsFIFO (
    const char * name,
    size_t objsize,
    size_t objn,
    unsigned objalign )
```

Creates a dynamic "objects FIFO" object.

Postcondition

A reference to the dynamic "objects FIFO" object is returned and the reference counter is initialized to one.
 The dynamic "objects FIFO" object is initialized and ready to use.

Parameters

in	<i>name</i>	name to be assigned to the new dynamic "objects FIFO" object
in	<i>objsize</i>	size of objects
in	<i>objn</i>	number of objects available
in	<i>objalign</i>	required objects alignment

Returns

The reference to the created dynamic "objects FIFO" object.

Return values

NULL	if the dynamic "objects FIFO" object cannot be allocated or a dynamic "objects FIFO" object with the same name exists.
-------------	--

Function Class:

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

7.52.4.16 chFactoryFindObjectSFIFO()

```
dyn_objects_fifo_t * chFactoryFindObjectSFIFO (
    const char * name )
```

Retrieves a dynamic "objects FIFO" object.

Postcondition

A reference to the dynamic "objects FIFO" object is returned with the reference counter increased by one.

Parameters

in	<i>name</i>	name of the dynamic "objects FIFO" object
----	-------------	---

Returns

The reference to the found dynamic "objects FIFO" object.

Return values

<code>NULL</code>	if a dynamic "objects FIFO" object with the specified name does not exist.
-------------------	--

Function Class:

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

7.52.4.17 chFactoryReleaseObjectsFIFO()

```
void chFactoryReleaseObjectsFIFO (
    dyn_objects_fifo_t * dofpx )
```

Releases a dynamic "objects FIFO" object.

The reference counter of the dynamic "objects FIFO" object is decreased by one, if reaches zero then the dynamic "objects FIFO" object memory is freed.

Parameters

in	<i>dofpx</i>	dynamic "objects FIFO" object reference
----	--------------	---

Function Class:

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

7.52.4.18 chFactoryCreatePipe()

```
dyn_pipe_t * chFactoryCreatePipe (
    const char * name,
    size_t size )
```

Creates a dynamic pipe object.

Postcondition

A reference to the dynamic pipe object is returned and the reference counter is initialized to one.

The dynamic pipe object is initialized and ready to use.

Parameters

in	<i>name</i>	name to be assigned to the new dynamic pipe object
in	<i>size</i>	pipe buffer size

Returns

The reference to the created dynamic pipe object.

Return values

<code>NULL</code>	if the dynamic pipe object cannot be allocated or a dynamic pipe object with the same name exists.
-------------------	--

Function Class:

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

7.52.4.19 chFactoryFindPipe()

```
dyn_pipe_t * chFactoryFindPipe (
    const char * name )
```

Retrieves a dynamic pipe object.

Postcondition

A reference to the dynamic pipe object is returned with the reference counter increased by one.

Parameters

<code>in</code>	<code>name</code>	name of the pipe object
-----------------	-------------------	-------------------------

Returns

The reference to the found dynamic pipe object.

Return values

<code>NULL</code>	if a dynamic pipe object with the specified name does not exist.
-------------------	--

Function Class:

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

7.52.4.20 chFactoryReleasePipe()

```
void chFactoryReleasePipe (
    dyn_pipe_t * dpp )
```

Releases a dynamic pipe object.

The reference counter of the dynamic pipe object is decreased by one, if reaches zero then the dynamic pipe object memory is freed.

Parameters

in	dpp	dynamic pipe object reference
----	-----	-------------------------------

Function Class:

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

7.52.4.21 chFactoryDuplicateReference()

```
static dyn_element_t* chFactoryDuplicateReference (
    dyn_element_t * dep ) [inline], [static]
```

Duplicates an object reference.

Note

This function can be used on any kind of dynamic object.

Parameters

in	dep	pointer to the element field of the object
----	-----	--

Returns

The duplicated object reference.

Function Class:

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

7.52.4.22 chFactoryGetObject()

```
static void* chFactoryGetObject (
    registered_object_t * rop ) [inline], [static]
```

Returns the pointer to the inner registered object.

Parameters

in	rop	registered object reference
----	-----	-----------------------------

Returns

The pointer to the registered object.

Function Class:

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

7.52.4.23 chFactoryGetBufferSize()

```
static size_t chFactoryGetBufferSize (
    dyn_buffer_t * dbp ) [inline], [static]
```

Returns the size of a generic dynamic buffer object.

Parameters

in	<i>dbp</i>	dynamic buffer object reference
----	------------	---------------------------------

Returns

The size of the buffer object in bytes.

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:

**7.52.4.24 chFactoryGetBuffer()**

```
static uint8_t* chFactoryGetBuffer (
    dyn_buffer_t * dbp ) [inline], [static]
```

Returns the pointer to the inner buffer.

Parameters

in	<i>dbp</i>	dynamic buffer object reference
----	------------	---------------------------------

Returns

The pointer to the dynamic buffer.

Function Class:

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

7.52.4.25 chFactoryGetSemaphore()

```
static semaphore_t* chFactoryGetSemaphore (
    dyn_semaphore_t * dsp ) [inline], [static]
```

Returns the pointer to the inner semaphore.

Parameters

in	<i>dsp</i>	dynamic semaphore object reference
----	------------	------------------------------------

Returns

The pointer to the semaphore.

Function Class:

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

7.52.4.26 chFactoryGetMailbox()

```
static mailbox_t* chFactoryGetMailbox (
    dyn_mailbox_t * dmp ) [inline], [static]
```

Returns the pointer to the inner mailbox.

Parameters

in	<i>dmp</i>	dynamic mailbox object reference
----	------------	----------------------------------

Returns

The pointer to the mailbox.

Function Class:

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

7.52.4.27 chFactoryGetObjectsFIFO()

```
static objects_fifo_t* chFactoryGetObjectsFIFO (
    dyn_objects_fifo_t * dofpx ) [inline], [static]
```

Returns the pointer to the inner objects FIFO.

Parameters

in	dofpx	dynamic "objects FIFO" object reference
----	-------	---

Returns

The pointer to the objects FIFO.

Function Class:

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

7.52.4.28 chFactoryGetPipe()

```
static pipe_t* chFactoryGetPipe (
    dyn_pipe_t * dpp ) [inline], [static]
```

Returns the pointer to the inner pipe.

Parameters

in	dpp	dynamic pipe object reference
----	-----	-------------------------------

Returns

The pointer to the pipe.

Function Class:

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

7.52.5 Variable Documentation

7.52.5.1 ch_factory

```
objects_factory_t ch_factory
```

Factory object static instance.

Note

It is a global object because it could be accessed through a specific debugger plugin.

Chapter 8

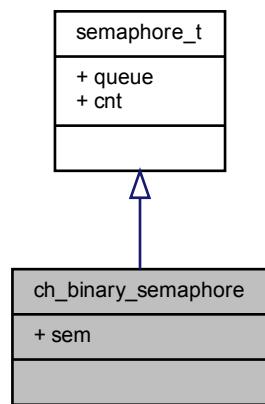
Data Structure Documentation

8.1 ch_binary_semaphore Struct Reference

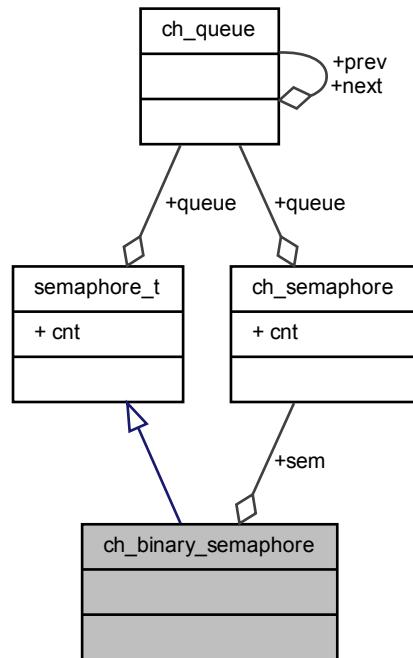
Binary semaphore type.

```
#include <chbsem.h>
```

Inheritance diagram for ch_binary_semaphore:



Collaboration diagram for ch_binary_semaphore:



Additional Inherited Members

8.1.1 Detailed Description

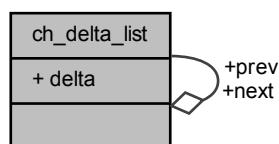
Binary semaphore type.

8.2 ch_delta_list Struct Reference

Delta list element and header structure.

```
#include <chlists.h>
```

Collaboration diagram for ch_delta_list:



Data Fields

- `ch_delta_list_t * next`

Next in the delta list.

- `ch_delta_list_t * prev`

Previous in the delta list.

- `sysinterval_t delta`

Time interval from previous.

8.2.1 Detailed Description

Delta list element and header structure.

8.2.2 Field Documentation

8.2.2.1 `next`

`ch_delta_list_t* ch_delta_list::next`

Next in the delta list.

8.2.2.2 `prev`

`ch_delta_list_t* ch_delta_list::prev`

Previous in the delta list.

8.2.2.3 `delta`

`sysinterval_t ch_delta_list::delta`

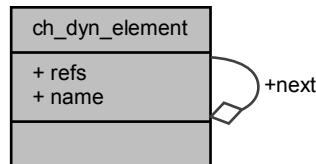
Time interval from previous.

8.3 ch_dyn_element Struct Reference

Type of a dynamic object list element.

```
#include <chfactory.h>
```

Collaboration diagram for ch_dyn_element:



Data Fields

- struct `ch_dyn_element` * `next`
Next dynamic object in the list.
- `ucnt_t refs`
Number of references to this object.

8.3.1 Detailed Description

Type of a dynamic object list element.

8.3.2 Field Documentation

8.3.2.1 next

```
struct ch_dyn_element* ch_dyn_element::next
```

Next dynamic object in the list.

8.3.2.2 refs

```
ucnt_t ch_dyn_element::refs
```

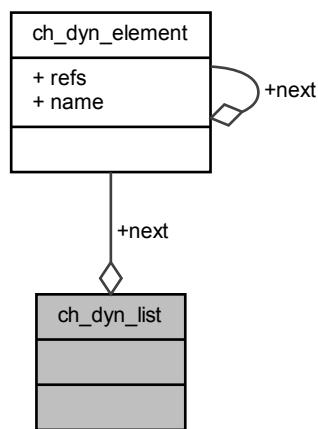
Number of references to this object.

8.4 ch_dyn_list Struct Reference

Type of a dynamic object list.

```
#include <chfactory.h>
```

Collaboration diagram for ch_dyn_list:



8.4.1 Detailed Description

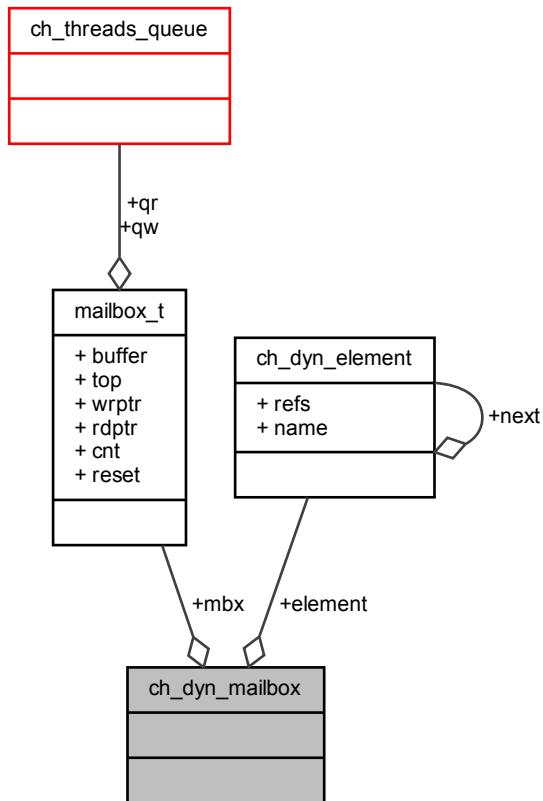
Type of a dynamic object list.

8.5 ch_dyn_mailbox Struct Reference

Type of a dynamic buffer object.

```
#include <chfactory.h>
```

Collaboration diagram for ch_dyn_mailbox:



Data Fields

- [dyn_element_t element](#)
List element of the dynamic buffer object.
- [mailbox_t mbx](#)
The mailbox.

8.5.1 Detailed Description

Type of a dynamic buffer object.

8.5.2 Field Documentation

8.5.2.1 element

```
dyn_element_t ch_dyn_mailbox::element
```

List element of the dynamic buffer object.

8.5.2.2 mbx

```
mailbox_t ch_dyn_mailbox::mbx
```

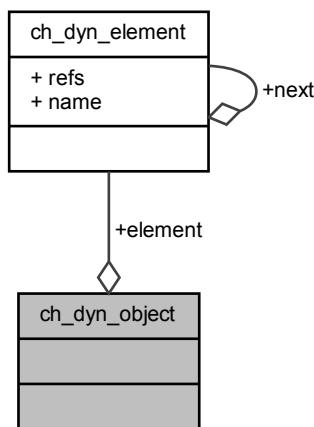
The mailbox.

8.6 ch_dyn_object Struct Reference

Type of a dynamic buffer object.

```
#include <chfactory.h>
```

Collaboration diagram for ch_dyn_object:



Data Fields

- `dyn_element_t element`

List element of the dynamic buffer object.

8.6.1 Detailed Description

Type of a dynamic buffer object.

8.6.2 Field Documentation

8.6.2.1 element

```
dyn_element_t ch_dyn_object::element
```

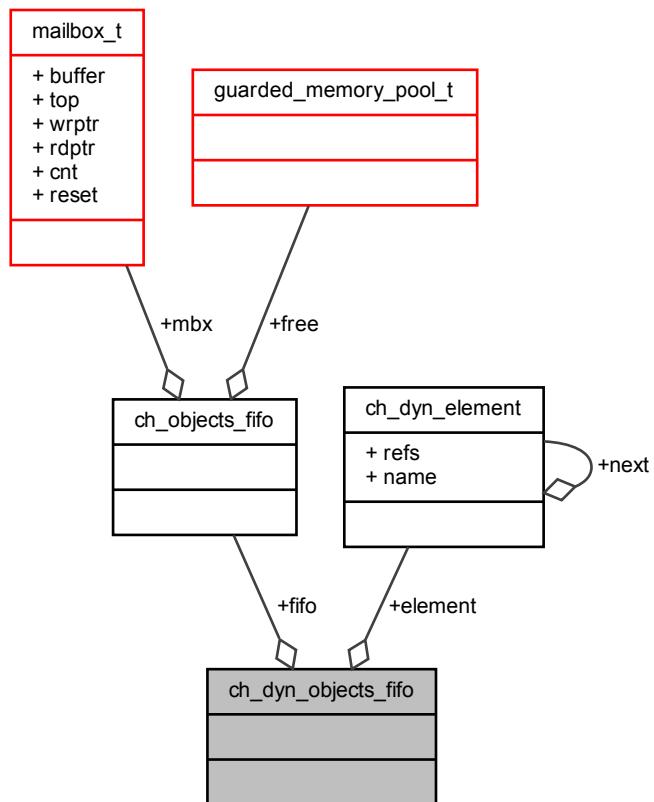
List element of the dynamic buffer object.

8.7 ch_dyn_objects_fifo Struct Reference

Type of a dynamic buffer object.

```
#include <chfactory.h>
```

Collaboration diagram for ch_dyn_objects_fifo:



Data Fields

- `dyn_element_t element`
List element of the dynamic buffer object.
- `objects_fifo_t fifo`
The objects FIFO.

8.7.1 Detailed Description

Type of a dynamic buffer object.

8.7.2 Field Documentation

8.7.2.1 element

`dyn_element_t ch_dyn_objects_fifo::element`

List element of the dynamic buffer object.

8.7.2.2 fifo

`objects_fifo_t ch_dyn_objects_fifo::fifo`

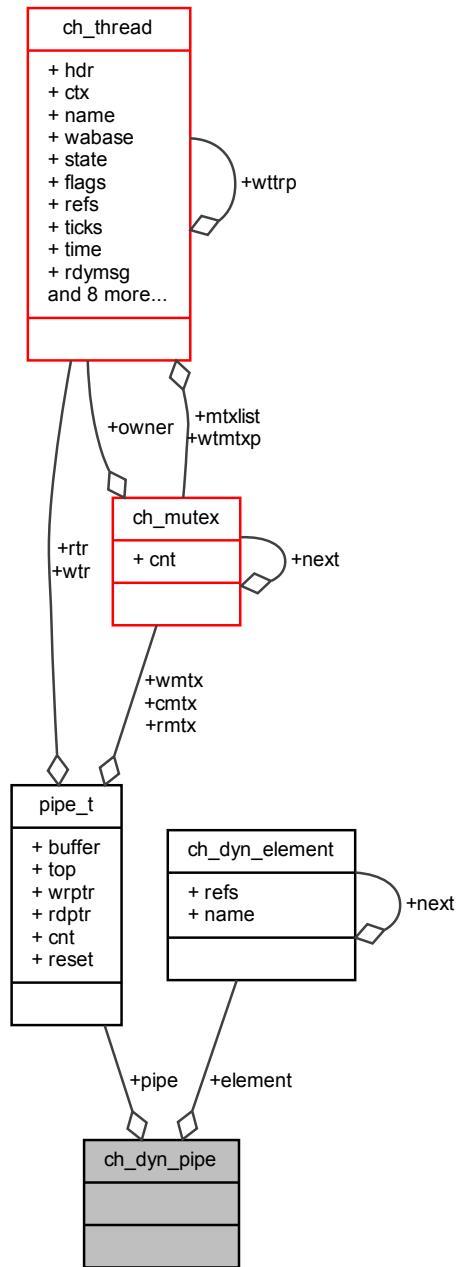
The objects FIFO.

8.8 ch_dyn_pipe Struct Reference

Type of a dynamic pipe object.

```
#include <chfactory.h>
```

Collaboration diagram for ch_dyn_pipe:



Data Fields

- `dyn_element_t element`

List element of the dynamic pipe object.

- `pipe_t pipe`

The pipe.

8.8.1 Detailed Description

Type of a dynamic pipe object.

8.8.2 Field Documentation

8.8.2.1 element

`dyn_element_t ch_dyn_pipe::element`

List element of the dynamic pipe object.

8.8.2.2 pipe

`pipe_t ch_dyn_pipe::pipe`

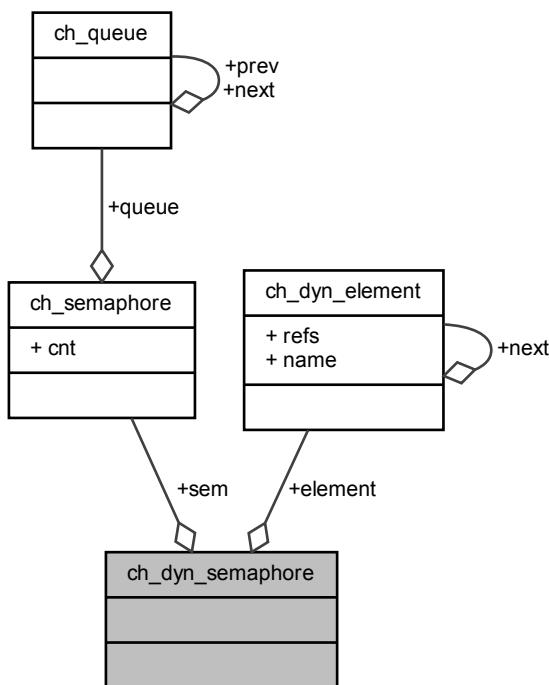
The pipe.

8.9 ch_dyn_semaphore Struct Reference

Type of a dynamic semaphore.

```
#include <chfactory.h>
```

Collaboration diagram for ch_dyn_semaphore:



Data Fields

- `dyn_element_t element`
List element of the dynamic semaphore.
- `semaphore_t sem`
The semaphore.

8.9.1 Detailed Description

Type of a dynamic semaphore.

8.9.2 Field Documentation

8.9.2.1 element

`dyn_element_t ch_dyn_semaphore::element`

List element of the dynamic semaphore.

8.9.2.2 sem

`semaphore_t ch_dyn_semaphore::sem`

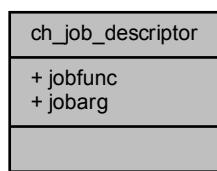
The semaphore.

8.10 ch_job_descriptor Struct Reference

Type of a job descriptor.

```
#include <chjobs.h>
```

Collaboration diagram for ch_job_descriptor:



Data Fields

- `job_function_t jobfunc`
Job function.
- `void * jobarg`
Argument to be passed to the job function.

8.10.1 Detailed Description

Type of a job descriptor.

8.10.2 Field Documentation

8.10.2.1 jobfunc

`job_function_t ch_job_descriptor::jobfunc`

Job function.

8.10.2.2 jobarg

`void* ch_job_descriptor::jobarg`

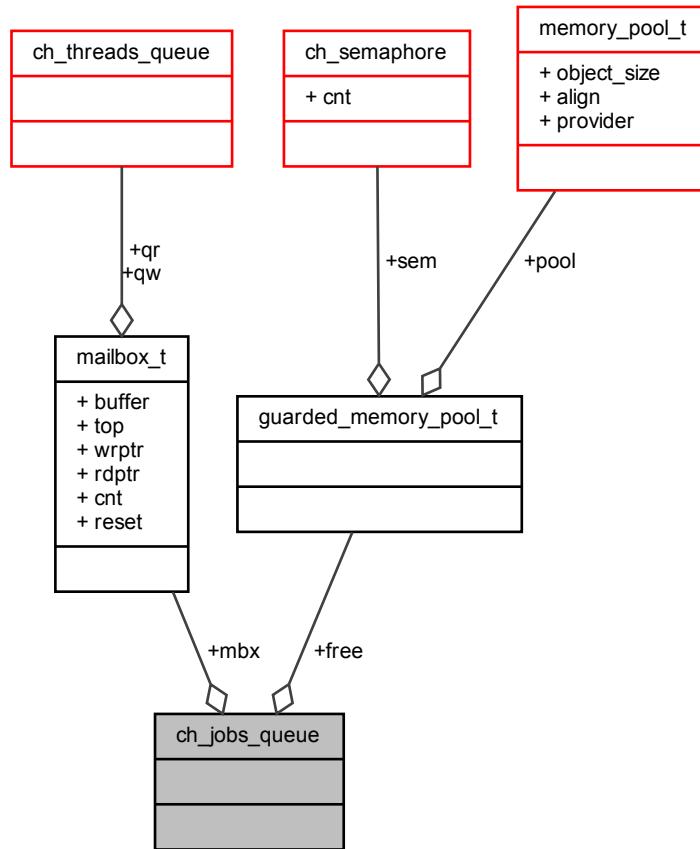
Argument to be passed to the job function.

8.11 ch_jobs_queue Struct Reference

Type of a jobs queue.

```
#include <chjobs.h>
```

Collaboration diagram for ch_jobs_queue:



Data Fields

- [guarded_memory_pool_t free](#)
Pool of the free jobs.
- [mailbox_t mbx](#)
Mailbox of the sent jobs.

8.11.1 Detailed Description

Type of a jobs queue.

8.11.2 Field Documentation

8.11.2.1 free

```
guarded_memory_pool_t ch_jobs_queue::free
```

Pool of the free jobs.

8.11.2.2 mbx

```
mailbox_t ch_jobs_queue::mbx
```

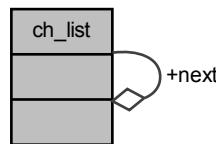
Mailbox of the sent jobs.

8.12 ch_list Struct Reference

Structure representing a generic single link list header and element.

```
#include <chlists.h>
```

Collaboration diagram for ch_list:



Data Fields

- `ch_list_t * next`

Next in the list/queue.

8.12.1 Detailed Description

Structure representing a generic single link list header and element.

8.12.2 Field Documentation

8.12.2.1 next

`ch_list_t* ch_list::next`

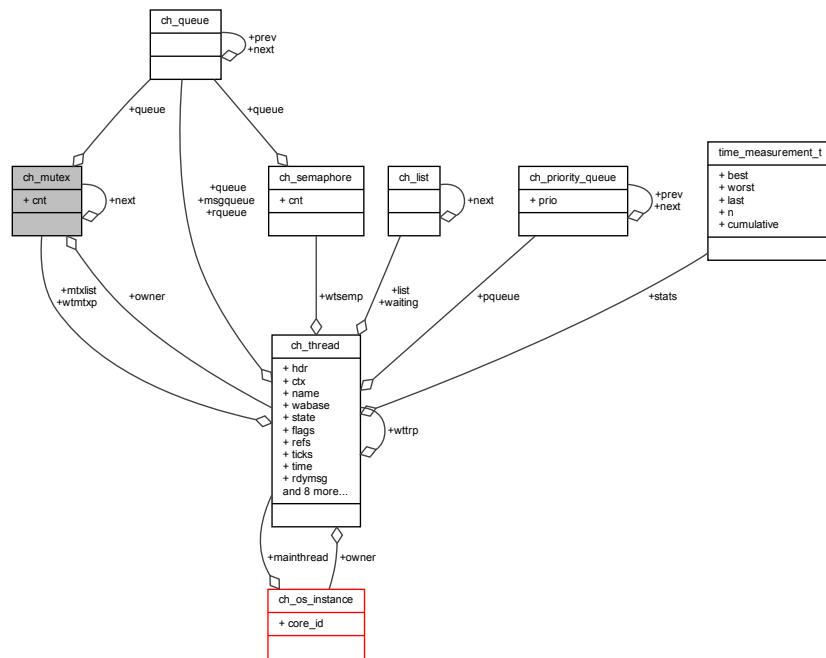
Next in the list/queue.

8.13 ch_mutex Struct Reference

Mutex structure.

```
#include <chmtx.h>
```

Collaboration diagram for ch_mutex:



Data Fields

- `ch_queue_t queue`

Queue of the threads sleeping on this mutex.

- `thread_t * owner`

Owner `thread_t` pointer or NULL.

- `mutex_t * next`

Next `mutex_t` into an owner-list or NULL.

- `cnt_t cnt`

Mutex recursion counter.

8.13.1 Detailed Description

Mutex structure.

8.13.2 Field Documentation

8.13.2.1 queue

`ch_queue_t ch_mutex::queue`

Queue of the threads sleeping on this mutex.

8.13.2.2 owner

`thread_t* ch_mutex::owner`

Owner `thread_t` pointer or `NULL`.

8.13.2.3 next

`mutex_t* ch_mutex::next`

Next `mutex_t` into an owner-list or `NULL`.

8.13.2.4 cnt

`cnt_t ch_mutex::cnt`

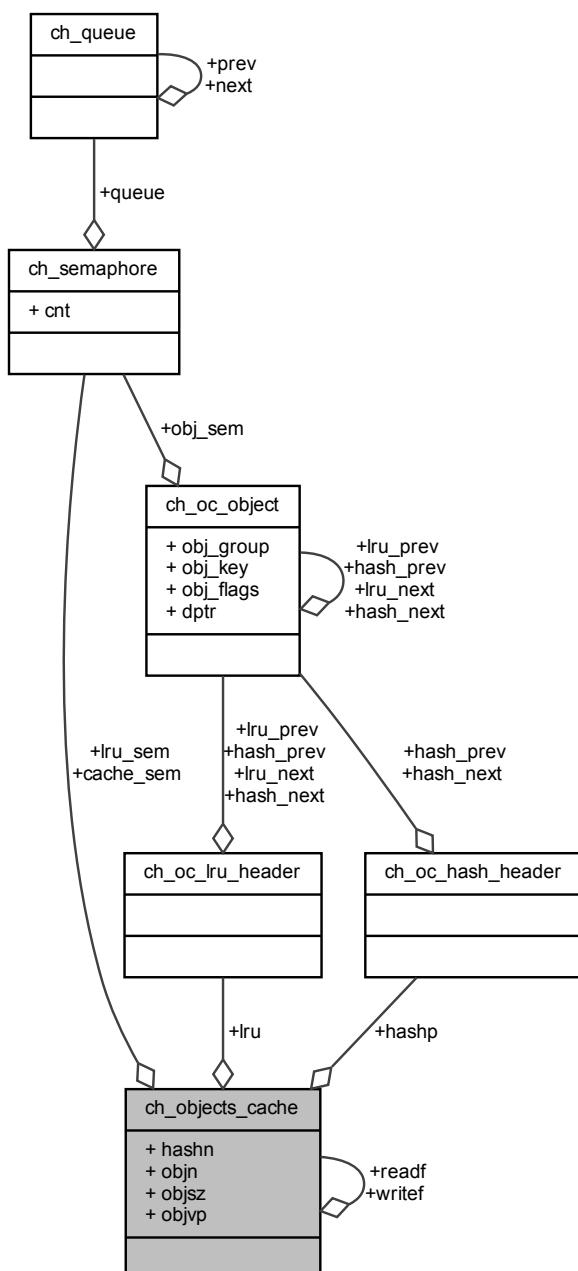
Mutex recursion counter.

8.14 ch_objects_cache Struct Reference

Structure representing a cache object.

```
#include <chobjcaches.h>
```

Collaboration diagram for ch_objects_cache:



Data Fields

- `ucnt_t hashn`
Number of elements in the hash table.
- `oc_hash_header_t * hashp`
Pointer to the hash table.
- `ucnt_t objn`
Number of elements in the objects table.
- `size_t objsz`
Size of elements in the objects table.
- `void * objvp`
Pointer to the objects table.
- `oc_lru_header_t lru`
LRU list header.
- `semaphore_t cache_sem`
Semaphore for cache access.
- `semaphore_t lru_sem`
Semaphore for LRU access.
- `oc_readf_t readf`
Reader functions for cached objects.
- `oc_writef_t writef`
Writer functions for cached objects.

8.14.1 Detailed Description

Structure representing a cache object.

8.14.2 Field Documentation

8.14.2.1 hashn

`ucnt_t ch_objects_cache::hashn`

Number of elements in the hash table.

8.14.2.2 hashp

`oc_hash_header_t* ch_objects_cache::hashp`

Pointer to the hash table.

8.14.2.3 objn

```
ucnt_t ch_objects_cache::objn
```

Number of elements in the objects table.

8.14.2.4 objsz

```
size_t ch_objects_cache::objsz
```

Size of elements in the objects table.

8.14.2.5 objvp

```
void* ch_objects_cache::objvp
```

Pointer to the objects table.

8.14.2.6 lru

```
oc_lru_header_t ch_objects_cache::lru
```

LRU list header.

8.14.2.7 cache_sem

```
semaphore_t ch_objects_cache::cache_sem
```

Semaphore for cache access.

8.14.2.8 lru_sem

```
semaphore_t ch_objects_cache::lru_sem
```

Semaphore for LRU access.

8.14.2.9 readf

```
oc_readf_t ch_objects_cache::readf
```

Reader functions for cached objects.

8.14.2.10 writef

```
oc_writef_t ch_objects_cache::writef
```

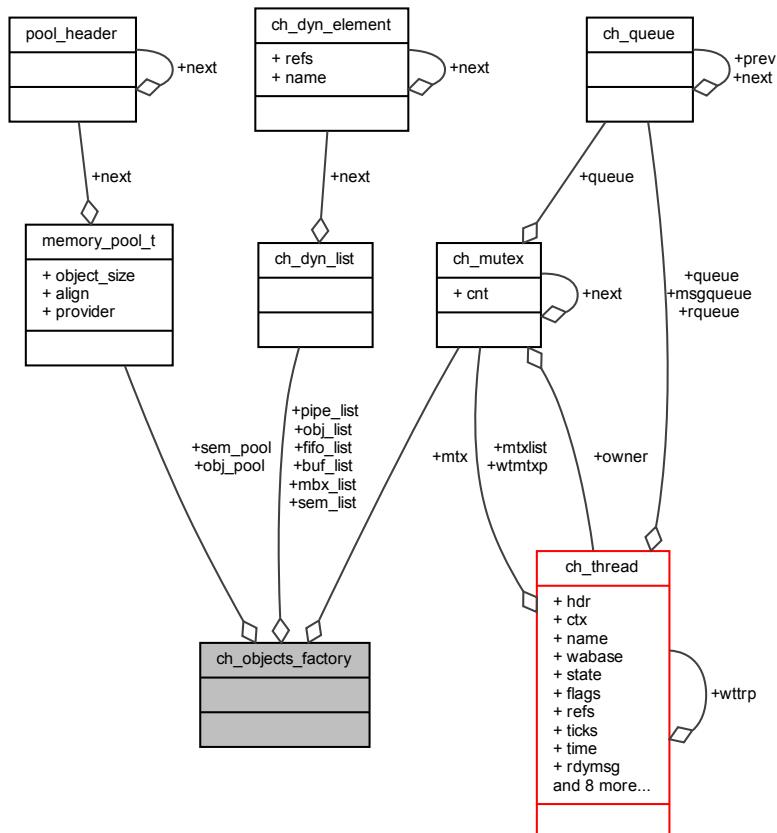
Writer functions for cached objects.

8.15 ch_objects_factory Struct Reference

Type of the factory main object.

```
#include <chfactory.h>
```

Collaboration diagram for ch_objects_factory:



Data Fields

- `mutex_t mtx`
Factory access mutex or semaphore.
- `dyn_list_t obj_list`
List of the registered objects.
- `memory_pool_t obj_pool`
Pool of the available registered objects.
- `dyn_list_t buf_list`
List of the allocated buffer objects.
- `dyn_list_t sem_list`
List of the allocated semaphores.
- `memory_pool_t sem_pool`
Pool of the available semaphores.
- `dyn_list_t mbx_list`
List of the allocated buffer objects.
- `dyn_list_t fifo_list`
List of the allocated "objects FIFO" objects.
- `dyn_list_t pipe_list`
List of the allocated pipe objects.

8.15.1 Detailed Description

Type of the factory main object.

8.15.2 Field Documentation

8.15.2.1 mtx

`mutex_t ch_objects_factory::mtx`

Factory access mutex or semaphore.

8.15.2.2 obj_list

`dyn_list_t ch_objects_factory::obj_list`

List of the registered objects.

8.15.2.3 obj_pool

```
memory_pool_t ch_objects_factory::obj_pool
```

Pool of the available registered objects.

8.15.2.4 buf_list

```
dyn_list_t ch_objects_factory::buf_list
```

List of the allocated buffer objects.

8.15.2.5 sem_list

```
dyn_list_t ch_objects_factory::sem_list
```

List of the allocated semaphores.

8.15.2.6 sem_pool

```
memory_pool_t ch_objects_factory::sem_pool
```

Pool of the available semaphores.

8.15.2.7 mbx_list

```
dyn_list_t ch_objects_factory::mbx_list
```

List of the allocated buffer objects.

8.15.2.8 fifo_list

```
dyn_list_t ch_objects_factory::fifo_list
```

List of the allocated "objects FIFO" objects.

8.15.2.9 pipe_list

`dyn_list_t ch_objects_factory::pipe_list`

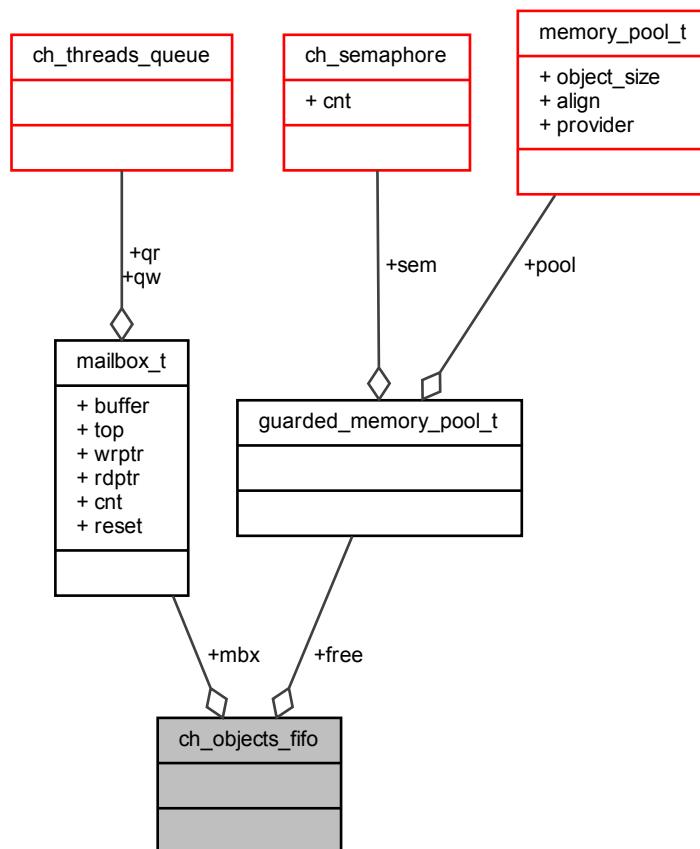
List of the allocated pipe objects.

8.16 ch_objects_fifo Struct Reference

Type of an objects FIFO.

```
#include <chobjfifo.h>
```

Collaboration diagram for ch_objects_fifo:



Data Fields

- `guarded_memory_pool_t free`

Pool of the free objects.

- `mailbox_t mbx`

Mailbox of the sent objects.

8.16.1 Detailed Description

Type of an objects FIFO.

8.16.2 Field Documentation

8.16.2.1 free

```
guarded_memory_pool_t ch_objects_fifo::free
```

Pool of the free objects.

8.16.2.2 mbx

```
mailbox_t ch_objects_fifo::mbx
```

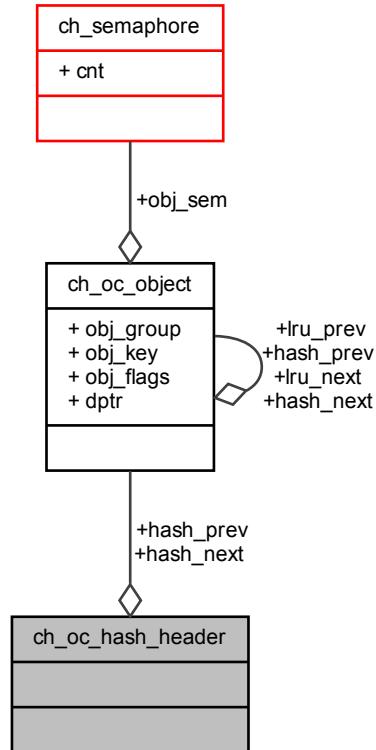
Mailbox of the sent objects.

8.17 ch_oc_hash_header Struct Reference

Structure representing an hash table element.

```
#include <chobjcaches.h>
```

Collaboration diagram for ch_oc_hash_header:



Data Fields

- `oc_object_t * hash_next`
Next in the collisions list.
- `oc_object_t * hash_prev`
Previous in the collisions list.

8.17.1 Detailed Description

Structure representing an hash table element.

8.17.2 Field Documentation

8.17.2.1 hash_next

`oc_object_t* ch_oc_hash_header::hash_next`

Next in the collisions list.

8.17.2.2 hash_prev

`oc_object_t* ch_oc_hash_header::hash_prev`

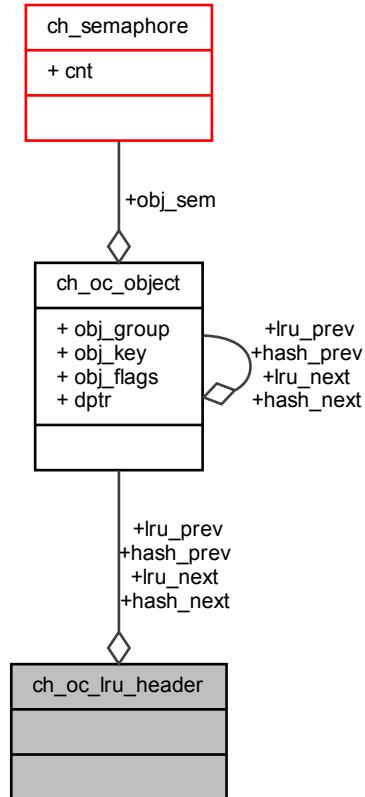
Previous in the collisions list.

8.18 ch_oc_lru_header Struct Reference

Structure representing an hash table element.

```
#include <chobjcaches.h>
```

Collaboration diagram for ch_oc_lru_header:



Data Fields

- `oc_object_t * hash_next`
Next in the collisions list.
- `oc_object_t * hash_prev`
Previous in the collisions list.
- `oc_object_t * lru_next`
Next in the LRU list.
- `oc_object_t * lru_prev`
Previous in the LRU list.

8.18.1 Detailed Description

Structure representing an hash table element.

8.18.2 Field Documentation

8.18.2.1 hash_next

`oc_object_t* ch_oc_lru_header::hash_next`

Next in the collisions list.

8.18.2.2 hash_prev

`oc_object_t* ch_oc_lru_header::hash_prev`

Previous in the collisions list.

8.18.2.3 lru_next

`oc_object_t* ch_oc_lru_header::lru_next`

Next in the LRU list.

8.18.2.4 lru_prev

`oc_object_t* ch_oc_lru_header::lru_prev`

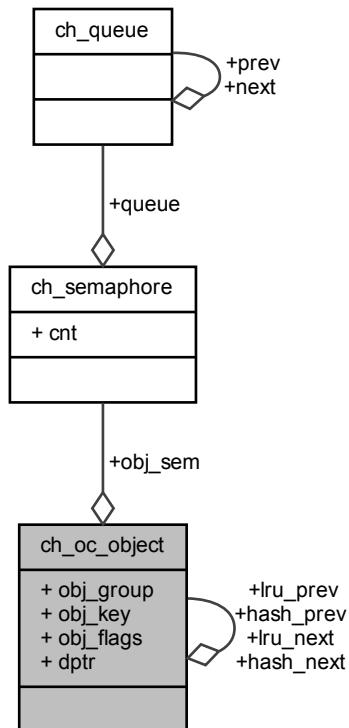
Previous in the LRU list.

8.19 ch_oc_object Struct Reference

Structure representing a cached object.

```
#include <chobjcaches.h>
```

Collaboration diagram for ch_oc_object:



Data Fields

- `oc_object_t * hash_next`
Next in the collisions list.
- `oc_object_t * hash_prev`
Previous in the collisions list.
- `oc_object_t * lru_next`
Next in the LRU list.
- `oc_object_t * lru_prev`
Previous in the LRU list.
- `uint32_t obj_group`
Object group.
- `uint32_t obj_key`
Object key.
- `semaphore_t obj_sem`

- *Semaphore for object access.*
- `oc_flags_t obj_flags`
 - *Object flags.*
- `void * dptr`
 - *User pointer.*

8.19.1 Detailed Description

Structure representing a cached object.

8.19.2 Field Documentation

8.19.2.1 hash_next

```
oc_object_t* ch_oc_object::hash_next
```

Next in the collisions list.

8.19.2.2 hash_prev

```
oc_object_t* ch_oc_object::hash_prev
```

Previous in the collisions list.

8.19.2.3 lru_next

```
oc_object_t* ch_oc_object::lru_next
```

Next in the LRU list.

8.19.2.4 lru_prev

```
oc_object_t* ch_oc_object::lru_prev
```

Previous in the LRU list.

8.19.2.5 obj_group

```
uint32_t ch_oc_object::obj_group
```

Object group.

8.19.2.6 obj_key

```
uint32_t ch_oc_object::obj_key
```

Object key.

8.19.2.7 obj_sem

```
semaphore_t ch_oc_object::obj_sem
```

Semaphore for object access.

8.19.2.8 obj_flags

```
oc_flags_t ch_oc_object::obj_flags
```

Object flags.

8.19.2.9 dptr

```
void* ch_oc_object::dptr
```

User pointer.

Note

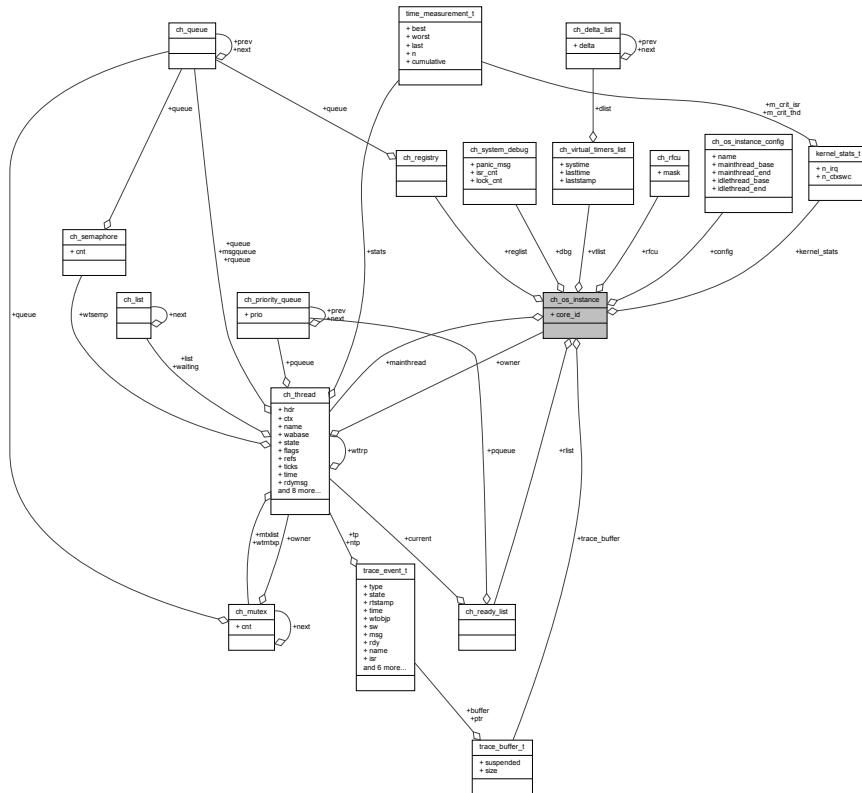
This pointer can be used to refer to external buffers, [chCacheObjectInit\(\)](#) initializes it to NULL.

8.20 ch_os_instance Struct Reference

System instance data structure.

```
#include <chobjects.h>
```

Collaboration diagram for ch_os_instance:



Data Fields

- **ready_list_t rlist**
Ready list header.
 - **virtual_timers_list_t vtlist**
Virtual timers delta list header.
 - **registry_t reglist**
Registry header.
 - **core_id_t core_id**
Core associated to this instance.
 - **rfcu_t rfcu**
Runtime Faults Collection Unit for this instance.
 - const **os_instance_config_t * config**
Pointer to the instance configuration data.
 - **thread_t mainthread**
Main thread descriptor.
 - **system_debug_t dbg**

- System debug.*
- `trace_buffer_t trace_buffer`
Trace buffer.
 - `kernel_stats_t kernel_stats`
Global kernel statistics.

8.20.1 Detailed Description

System instance data structure.

8.20.2 Field Documentation

8.20.2.1 rlist

```
ready_list_t ch_os_instance::rlist
```

Ready list header.

8.20.2.2 vtlist

```
virtual_timers_list_t ch_os_instance::vtlist
```

Virtual timers delta list header.

8.20.2.3 reglist

```
registry_t ch_os_instance::reglist
```

Registry header.

Note

This field is present only if the SMP mode is disabled.

8.20.2.4 core_id

```
core_id_t ch_os_instance::core_id
```

Core associated to this instance.

8.20.2.5 rfcu

```
rfcu_t ch_os_instance::rfcu
```

Runtime Faults Collection Unit for this instance.

Note

This field is present only if the SMP mode is disabled.

8.20.2.6 config

```
const os_instance_config_t* ch_os_instance::config
```

Pointer to the instance configuration data.

8.20.2.7 mainthread

```
thread_t ch_os_instance::mainthread
```

Main thread descriptor.

8.20.2.8 dbg

```
system_debug_t ch_os_instance::dbg
```

System debug.

8.20.2.9 trace_buffer

```
trace_buffer_t ch_os_instance::trace_buffer
```

Trace buffer.

8.20.2.10 kernel_stats

```
kernel_stats_t ch_os_instance::kernel_stats
```

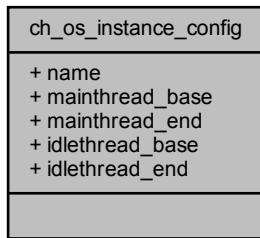
Global kernel statistics.

8.21 ch_os_instance_config Struct Reference

Type of an system instance configuration.

```
#include <chobjects.h>
```

Collaboration diagram for ch_os_instance_config:



Data Fields

- const char * **name**
Instance name.
- **stkalign_t * mainthread_base**
Lower limit of the main function thread stack.
- **stkalign_t * mainthread_end**
Upper limit of the main function thread stack.
- **stkalign_t * idlethread_base**
Lower limit of the dedicated idle thread stack.
- **stkalign_t * idlethread_end**
Upper limit of the dedicated idle thread stack.

8.21.1 Detailed Description

Type of an system instance configuration.

8.21.2 Field Documentation

8.21.2.1 name

```
const char* ch_os_instance_config::name
```

Instance name.

8.21.2.2 mainthread_base

```
stkalign_t* ch_os_instance_config::mainthread_base
```

Lower limit of the main function thread stack.

8.21.2.3 mainthread_end

```
stkalign_t* ch_os_instance_config::mainthread_end
```

Upper limit of the main function thread stack.

8.21.2.4 idlethread_base

```
stkalign_t* ch_os_instance_config::idlethread_base
```

Lower limit of the dedicated idle thread stack.

8.21.2.5 idlethread_end

```
stkalign_t* ch_os_instance_config::idlethread_end
```

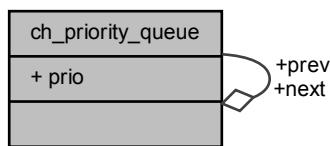
Upper limit of the dedicated idle thread stack.

8.22 ch_priority_queue Struct Reference

Structure representing a generic priority-ordered bidirectional linked list header and element.

```
#include <chlists.h>
```

Collaboration diagram for ch_priority_queue:



Data Fields

- `ch_priority_queue_t * next`

Next in the queue.

- `ch_priority_queue_t * prev`

Previous in the queue.

- `tprio_t prio`

Priority of this element.

8.22.1 Detailed Description

Structure representing a generic priority-ordered bidirectional linked list header and element.

Note

Link fields are void pointers in order to avoid aliasing issues.

8.22.2 Field Documentation

8.22.2.1 `next`

`ch_priority_queue_t* ch_priority_queue::next`

Next in the queue.

8.22.2.2 `prev`

`ch_priority_queue_t* ch_priority_queue::prev`

Previous in the queue.

8.22.2.3 `prio`

`tprio_t ch_priority_queue::prio`

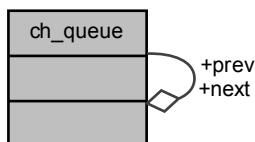
Priority of this element.

8.23 ch_queue Struct Reference

Structure representing a generic bidirectional linked list header and element.

```
#include <chlists.h>
```

Collaboration diagram for ch_queue:



Data Fields

- [ch_queue_t * next](#)
Next in the list/queue.
- [ch_queue_t * prev](#)
Previous in the queue.

8.23.1 Detailed Description

Structure representing a generic bidirectional linked list header and element.

8.23.2 Field Documentation

8.23.2.1 next

```
ch_queue_t* ch_queue::next
```

Next in the list/queue.

8.23.2.2 prev

`ch_queue_t* ch_queue::prev`

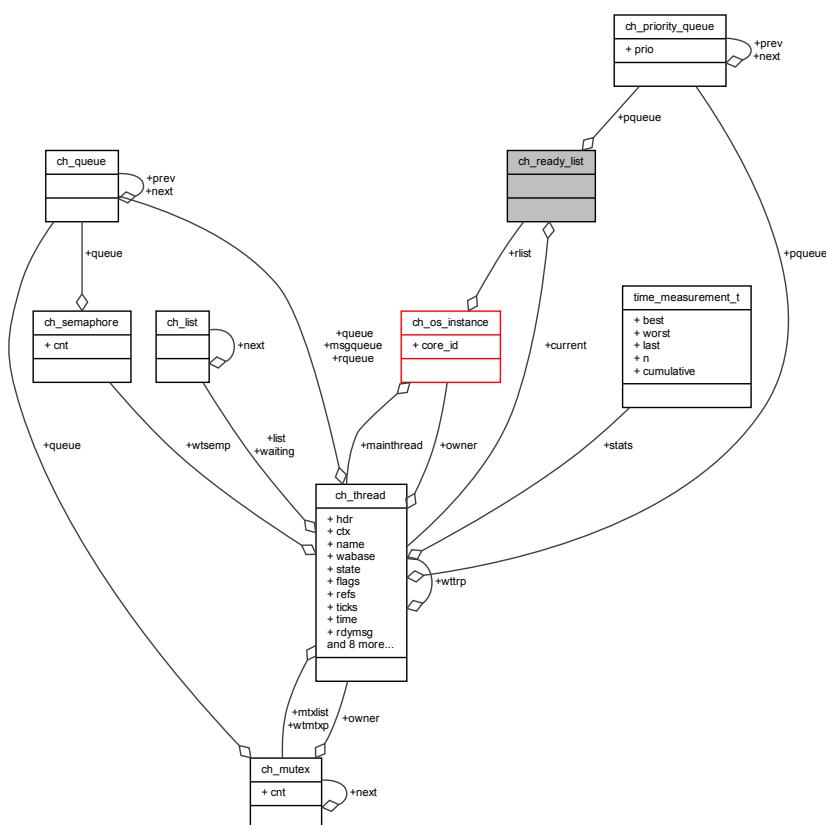
Previous in the queue.

8.24 ch_ready_list Struct Reference

Type of a ready list header.

```
#include <chobjects.h>
```

Collaboration diagram for ch_ready_list:



Data Fields

- `ch_priority_queue_t pqueue`

Threads ordered queues header.

- `thread_t * current`

The currently running thread.

8.24.1 Detailed Description

Type of a ready list header.

8.24.2 Field Documentation

8.24.2.1 pqueue

```
ch_priority_queue_t ch_ready_list::pqueue
```

Threads ordered queues header.

Note

The priority field must be initialized to zero.

8.24.2.2 current

```
thread_t* ch_ready_list::current
```

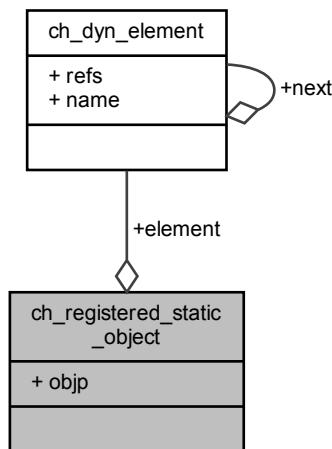
The currently running thread.

8.25 ch_registered_static_object Struct Reference

Type of a registered object.

```
#include <chfactory.h>
```

Collaboration diagram for ch_registered_static_object:



Data Fields

- `dyn_element_t element`
List element of the registered object.

- `void * objp`
Pointer to the object.

8.25.1 Detailed Description

Type of a registered object.

8.25.2 Field Documentation

8.25.2.1 element

```
dyn_element_t ch_registered_static_object::element
```

List element of the registered object.

8.25.2.2 objp

```
void* ch_registered_static_object::objp
```

Pointer to the object.

Note

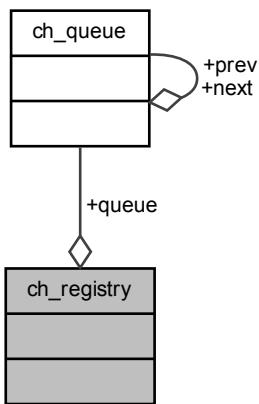
The type of the object is not stored in anyway.

8.26 ch_registry Struct Reference

Type of a registry structure.

```
#include <chobjects.h>
```

Collaboration diagram for ch_registry:



Data Fields

- [ch_queue_t queue](#)
Registry queue header.

8.26.1 Detailed Description

Type of a registry structure.

8.26.2 Field Documentation

8.26.2.1 queue

[ch_queue_t](#) ch_registry::queue

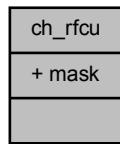
Registry queue header.

8.27 ch_rfcu Struct Reference

Type of an RFCU structure.

```
#include <chrfcu.h>
```

Collaboration diagram for ch_rfcu:



Data Fields

- [rfcu_mask_t mask](#)

Mask of the pending runtime faults.

8.27.1 Detailed Description

Type of an RFCU structure.

8.27.2 Field Documentation

8.27.2.1 mask

```
rfcu_mask_t ch_rfcu::mask
```

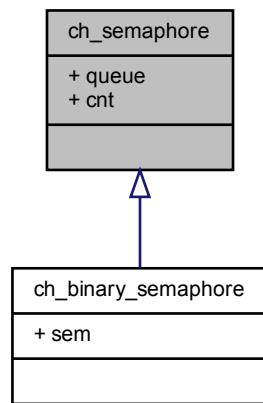
Mask of the pending runtime faults.

8.28 ch_semaphore Struct Reference

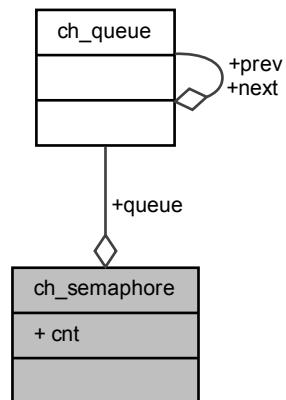
Semaphore structure.

```
#include <chsem.h>
```

Inheritance diagram for ch_semaphore:



Collaboration diagram for ch_semaphore:



Data Fields

- [ch_queue_t queue](#)

Queue of the threads sleeping on this semaphore.

- `cnt_t cnt`

The semaphore counter.

8.28.1 Detailed Description

Semaphore structure.

8.28.2 Field Documentation

8.28.2.1 queue

`ch_queue_t ch_semaphore::queue`

Queue of the threads sleeping on this semaphore.

8.28.2.2 cnt

`cnt_t ch_semaphore::cnt`

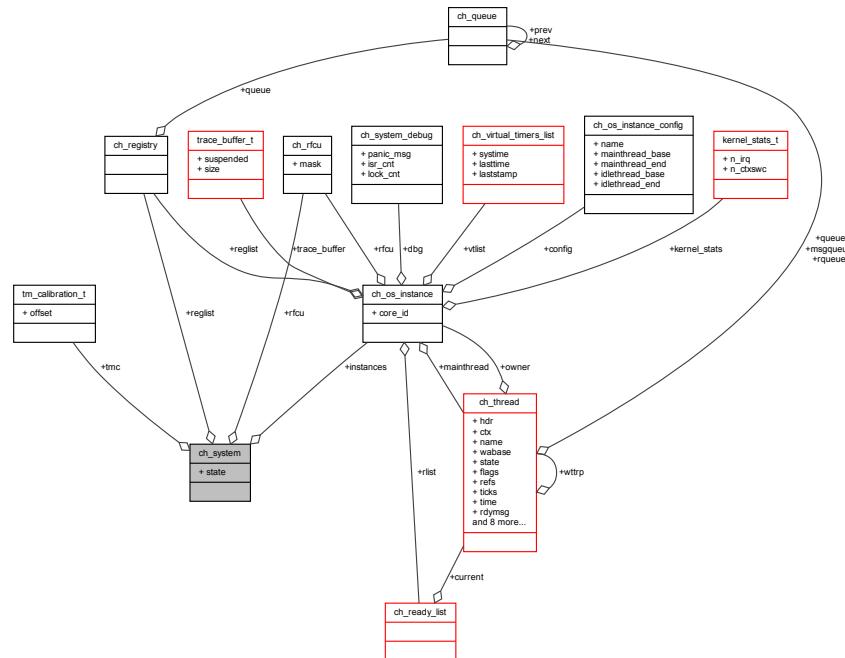
The semaphore counter.

8.29 ch_system Struct Reference

Type of system data structure.

```
#include <chobjects.h>
```

Collaboration diagram for ch_system:



Data Fields

- **system_state_t state**
Operating system state.
- **os_instance_t * instances [PORT_CORES_NUMBER]**
Initialized OS instances or NULL.
- **tm_calibration_t tmc**
Time measurement calibration data.
- **registry_t reglist**
Registry header.
- **rfcu_t rfcu**
Runtime Faults Collection Unit.

8.29.1 Detailed Description

Type of system data structure.

8.29.2 Field Documentation

8.29.2.1 state

```
system_state_t ch_system::state
```

Operating system state.

8.29.2.2 instances

```
os_instance_t* ch_system::instances[PORT_CORES_NUMBER]
```

Initialized OS instances or NULL.

8.29.2.3 tmc

```
tm_calibration_t ch_system::tmc
```

Time measurement calibration data.

8.29.2.4 reglist

```
registry_t ch_system::reglist
```

Registry header.

Note

This field is present only if the SMP mode is enabled.

8.29.2.5 rfcu

```
rfcu_t ch_system::rfcu
```

Runtime Faults Collection Unit.

Note

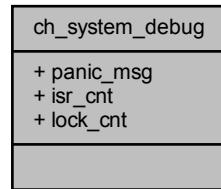
This field is present only if the SMP mode is enabled.

8.30 ch_system_debug Struct Reference

System debug data structure.

```
#include <chdebug.h>
```

Collaboration diagram for ch_system_debug:



Data Fields

- const char *volatile [panic_msg](#)
Pointer to the panic message.
- [cnt_t isr_cnt](#)
ISR nesting level.
- [cnt_t lock_cnt](#)
Lock nesting level.

8.30.1 Detailed Description

System debug data structure.

8.30.2 Field Documentation

8.30.2.1 [panic_msg](#)

```
const char* volatile ch_system_debug::panic_msg
```

Pointer to the panic message.

This pointer is meant to be accessed through the debugger, it is written once and then the system is halted.

Note

Accesses to this pointer must never be optimized out so the field itself is declared volatile.

8.30.2.2 isr_cnt

`cnt_t ch_system_debug::isr_cnt`

ISR nesting level.

8.30.2.3 lock_cnt

`cnt_t ch_system_debug::lock_cnt`

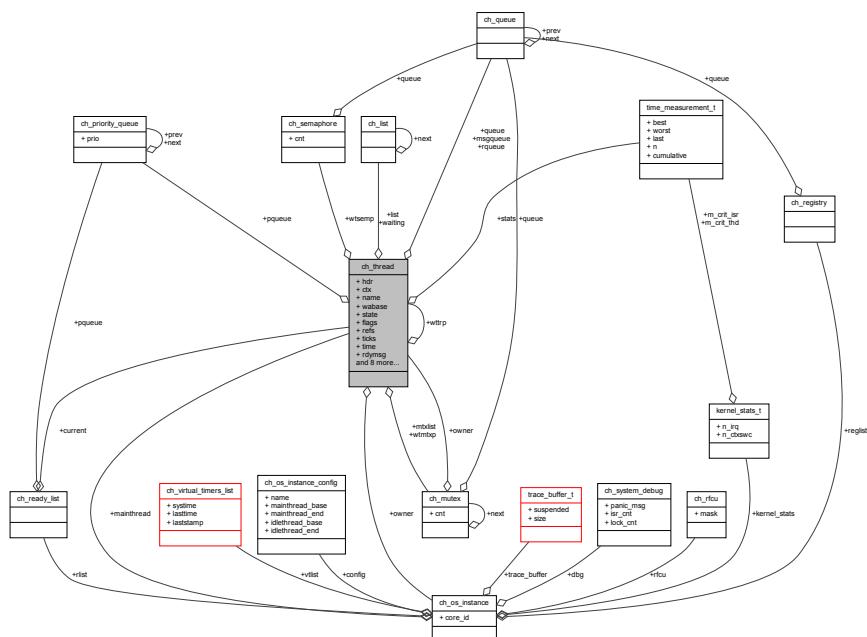
Lock nesting level.

8.31 ch_thread Struct Reference

Structure representing a thread.

```
#include <chobjects.h>
```

Collaboration diagram for ch_thread:



Data Fields

- union {
 - `ch_list_t list`
Threads lists element.
 - `ch_queue_t queue`
Threads queues element.
 - `ch_priority_queue_t pqueue`
Threads ordered queues element.
} `hdr`

- Shared list headers.
- struct `port_context` `ctx`
Processor context.
- `ch_queue_t rqueue`
Registry queue element.
- `os_instance_t * owner`
OS instance owner of this thread.
- const char * `name`
Thread name or NULL.
- `stkalign_t * wabase`
Working area base address.
- `tstate_t state`
Current thread state.
- `tmode_t flags`
Various thread flags.
- `trefs_t refs`
References to this thread.
- `tslices_t ticks`
Number of ticks remaining to this thread.
- volatile `systime_t time`
Thread consumed time in ticks.
- union {
 - `msg_t rdymsg`
Thread wakeup code.
 - `msg_t exitcode`
Thread exit code.
 - void * `wtobjp`
Pointer to a generic "wait" object.
 - `thread_reference_t * wttrp`
Pointer to a generic thread reference object.
 - `msg_t sentmsg`
Thread sent message.
 - struct `ch_semaphore` * `wtsemp`
Pointer to a generic semaphore object.
 - struct `ch_mutex` * `wtmtx`
Pointer to a generic mutex object.
 - `eventmask_t ewmask`
Enabled events mask.
} `u`

- State-specific fields.
- `ch_list_t waiting`
Termination waiting list.

- `ch_queue_t msgqueue`
Messages queue.
- `eventmask_t pending`
Pending events mask.
- struct `ch_mutex * mtxlist`
List of the mutexes owned by this thread.
- `tprio_t realprio`
Thread's own, non-inherited, priority.
- `void * mpool`
Memory Pool where the thread workspace is returned.
- `time_measurement_t stats`
Thread statistics.

8.31.1 Detailed Description

Structure representing a thread.

Note

Not all the listed fields are always needed, by switching off some not needed ChibiOS/RT subsystems it is possible to save RAM space by shrinking this structure.

8.31.2 Field Documentation

8.31.2.1 list

`ch_list_t ch_thread::list`

Threads lists element.

8.31.2.2 queue

`ch_queue_t ch_thread::queue`

Threads queues element.

8.31.2.3 pqueue

`ch_priority_queue_t ch_thread::pqueue`

Threads ordered queues element.

8.31.2.4 hdr

```
union { ... } ch_thread::hdr
```

Shared list headers.

8.31.2.5 ctx

```
struct port_context ch_thread::ctx
```

Processor context.

8.31.2.6 rqueue

```
ch_queue_t ch_thread::rqueue
```

Registry queue element.

8.31.2.7 owner

```
os_instance_t* ch_thread::owner
```

OS instance owner of this thread.

8.31.2.8 name

```
const char* ch_thread::name
```

Thread name or NULL.

8.31.2.9 wabase

```
stkalign_t* ch_thread::wabase
```

Working area base address.

Note

This pointer is used for stack overflow checks and for dynamic threading.

8.31.2.10 state

```
tstate_t ch_thread::state
```

Current thread state.

8.31.2.11 flags

```
tmode_t ch_thread::flags
```

Various thread flags.

8.31.2.12 refs

```
trefs_t ch_thread::refs
```

References to this thread.

8.31.2.13 ticks

```
tslices_t ch_thread::ticks
```

Number of ticks remaining to this thread.

8.31.2.14 time

```
volatile systime_t ch_thread::time
```

Thread consumed time in ticks.

Note

This field can overflow.

8.31.2.15 rdymsg

```
msg_t ch_thread::rdymsg
```

Thread wakeup code.

Note

This field contains the low level message sent to the thread by the waking thread or interrupt handler. The value is valid after exiting the [chSchWakeupS\(\)](#) function.

8.31.2.16 exitcode

```
msg_t ch_thread::exitcode
```

Thread exit code.

Note

The thread termination code is stored in this field in order to be retrieved by the thread performing a [chThdWait\(\)](#) on this thread.

8.31.2.17 wtobjp

```
void* ch_thread::wtobjp
```

Pointer to a generic "wait" object.

Note

This field is used to get a generic pointer to a synchronization object and is valid when the thread is in one of the wait states.

8.31.2.18 wttrp

```
thread_reference_t* ch_thread::wttrp
```

Pointer to a generic thread reference object.

Note

This field is used to get a pointer to a synchronization object and is valid when the thread is in `CH_STATE←_SUSPENDED` state.

8.31.2.19 sentmsg

```
msg_t ch_thread::sentmsg
```

Thread sent message.

8.31.2.20 wtsemp

```
struct ch_semaphore* ch_thread::wtsemp
```

Pointer to a generic semaphore object.

Note

This field is used to get a pointer to a synchronization object and is valid when the thread is in CH_STATE↔_WTSEM state.

8.31.2.21 wtmtxp

```
struct ch_mutex* ch_thread::wtmtxp
```

Pointer to a generic mutex object.

Note

This field is used to get a pointer to a synchronization object and is valid when the thread is in CH_STATE↔_WTMTX state.

8.31.2.22 ewmask

```
eventmask_t ch_thread::ewmask
```

Enabled events mask.

Note

This field is only valid while the thread is in the CH_STATE_WTOREVT or CH_STATE_WTANDEVT states.

8.31.2.23 u

```
union { ... } ch_thread::u
```

State-specific fields.

Note

All the fields declared in this union are only valid in the specified state or condition and are thus volatile.

8.31.2.24 waiting

```
ch_list_t ch_thread::waiting
```

Termination waiting list.

8.31.2.25 msgqueue

```
ch_queue_t ch_thread::msgqueue
```

Messages queue.

8.31.2.26 epending

```
eventmask_t ch_thread::epending
```

Pending events mask.

8.31.2.27 mtxlist

```
struct ch_mutex* ch_thread::mtxlist
```

List of the mutexes owned by this thread.

Note

The list is terminated by a NULL in this field.

8.31.2.28 realprio

`tprio_t ch_thread::realprio`

Thread's own, non-inherited, priority.

8.31.2.29 mpool

`void* ch_thread::mpool`

Memory Pool where the thread workspace is returned.

8.31.2.30 stats

`time_measurement_t ch_thread::stats`

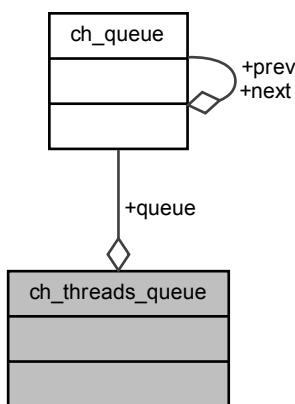
Thread statistics.

8.32 ch_threads_queue Struct Reference

Type of a threads queue.

```
#include <chobjects.h>
```

Collaboration diagram for ch_threads_queue:



Data Fields

- `ch_queue_t queue`
Threads queue header.

8.32.1 Detailed Description

Type of a threads queue.

8.32.2 Field Documentation

8.32.2.1 queue

`ch_queue_t ch_threads_queue::queue`

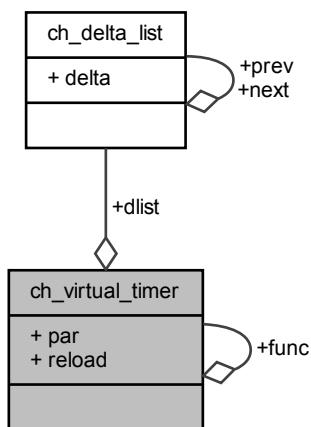
Threads queue header.

8.33 ch_virtual_timer Struct Reference

Structure representing a Virtual Timer.

```
#include <chobjects.h>
```

Collaboration diagram for `ch_virtual_timer`:



Data Fields

- `ch_delta_list_t dlist`
Delta list element.
- `vfunc_t func`
Timer callback function pointer.
- `void * par`
Timer callback function parameter.
- `sysinterval_t reload`
Current reload interval.

8.33.1 Detailed Description

Structure representing a Virtual Timer.

8.33.2 Field Documentation

8.33.2.1 dlist

`ch_delta_list_t ch_virtual_timer::dlist`

Delta list element.

8.33.2.2 func

`vfunc_t ch_virtual_timer::func`

Timer callback function pointer.

8.33.2.3 par

`void* ch_virtual_timer::par`

Timer callback function parameter.

8.33.2.4 reload

`sysinterval_t ch_virtual_timer::reload`

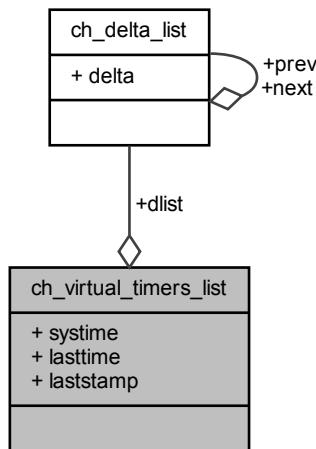
Current reload interval.

8.34 ch_virtual_timers_list Struct Reference

Type of virtual timers list header.

```
#include <chobjects.h>
```

Collaboration diagram for ch_virtual_timers_list:



Data Fields

- [ch_delta_list_t](#) `dlist`
Delta list header.
- volatile [systime_t](#) `systime`
System Time counter.
- [systime_t](#) `lasttime`
System time of the last tick event.
- volatile [uint64_t](#) `laststamp`
Last generated time stamp.

8.34.1 Detailed Description

Type of virtual timers list header.

Note

The timers list is implemented as a double link bidirectional list in order to make the unlink time constant, the reset of a virtual timer is often used in the code.

8.34.2 Field Documentation

8.34.2.1 dlist

```
ch_delta_list_t ch_virtual_timers_list::dlist
```

Delta list header.

8.34.2.2 systime

```
volatile systime_t ch_virtual_timers_list::systime
```

System Time counter.

8.34.2.3 lasttime

```
systime_t ch_virtual_timers_list::lasttime
```

System time of the last tick event.

8.34.2.4 laststamp

```
volatile uint64_t ch_virtual_timers_list::laststamp
```

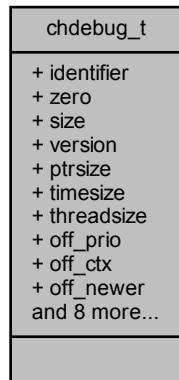
Last generated time stamp.

8.35 chdebug_t Struct Reference

ChibiOS/RT memory signature record.

```
#include <chregistry.h>
```

Collaboration diagram for chdebug_t:



Data Fields

- char **identifier** [4]
Always set to "main".
- uint8_t **zero**
Must be zero.
- uint8_t **size**
Size of this structure.
- uint16_t **version**
Encoded ChibiOS/RT version.
- uint8_t **ptrsize**
Size of a pointer.
- uint8_t **timesize**
Size of a systime_t.
- uint8_t **threadsize**
Size of a thread_t.
- uint8_t **off_prio**
Offset of prio field.
- uint8_t **off_ctx**

Offset of ctx field.

- uint8_t off_newer
Offset of newer field.
- uint8_t off_older
Offset of older field.
- uint8_t off_name
Offset of name field.
- uint8_t off_stklimit
Offset of stklimit field.
- uint8_t off_state
Offset of state field.
- uint8_t off_flags
Offset of flags field.
- uint8_t off_refs
Offset of refs field.
- uint8_t off_preempt
Offset of preempt field.
- uint8_t off_time
Offset of time field.

8.35.1 Detailed Description

ChibiOS/RT memory signature record.

8.35.2 Field Documentation

8.35.2.1 identifier

```
char chdebug_t::identifier[4]
```

Always set to "main".

8.35.2.2 zero

```
uint8_t chdebug_t::zero
```

Must be zero.

8.35.2.3 size

```
uint8_t chdebug_t::size
```

Size of this structure.

8.35.2.4 version

```
uint16_t chdebug_t::version
```

Encoded ChibiOS/RT version.

8.35.2.5 ptrsize

```
uint8_t chdebug_t::ptrsize
```

Size of a pointer.

8.35.2.6 timesize

```
uint8_t chdebug_t::timesize
```

Size of a systime_t.

8.35.2.7 threadsize

```
uint8_t chdebug_t::threadsize
```

Size of a thread_t.

8.35.2.8 off_prio

```
uint8_t chdebug_t::off_prio
```

Offset of prio field.

8.35.2.9 off_ctxt

```
uint8_t chdebug_t::off_ctxt
```

Offset of `ctxt` field.

8.35.2.10 off_newer

```
uint8_t chdebug_t::off_newer
```

Offset of `newer` field.

8.35.2.11 off_older

```
uint8_t chdebug_t::off_older
```

Offset of `older` field.

8.35.2.12 off_name

```
uint8_t chdebug_t::off_name
```

Offset of `name` field.

8.35.2.13 off_stklimit

```
uint8_t chdebug_t::off_stklimit
```

Offset of `stklimit` field.

8.35.2.14 off_state

```
uint8_t chdebug_t::off_state
```

Offset of `state` field.

8.35.2.15 off_flags

`uint8_t chdebug_t::off_flags`

Offset of `flags` field.

8.35.2.16 off_refs

`uint8_t chdebug_t::off_refs`

Offset of `refs` field.

8.35.2.17 off_preempt

`uint8_t chdebug_t::off_preempt`

Offset of `preempt` field.

8.35.2.18 off_time

`uint8_t chdebug_t::off_time`

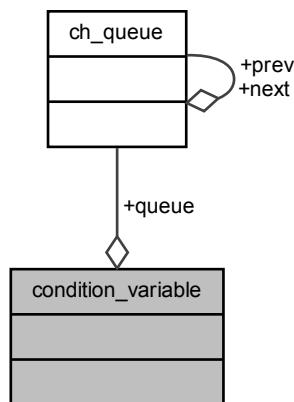
Offset of `time` field.

8.36 condition_variable Struct Reference

`condition_variable_t` structure.

#include <chcond.h>

Collaboration diagram for `condition_variable`:



Data Fields

- `ch_queue_t queue`

Condition variable threads queue.

8.36.1 Detailed Description

`condition_variable_t` structure.

8.36.2 Field Documentation

8.36.2.1 queue

`ch_queue_t condition_variable::queue`

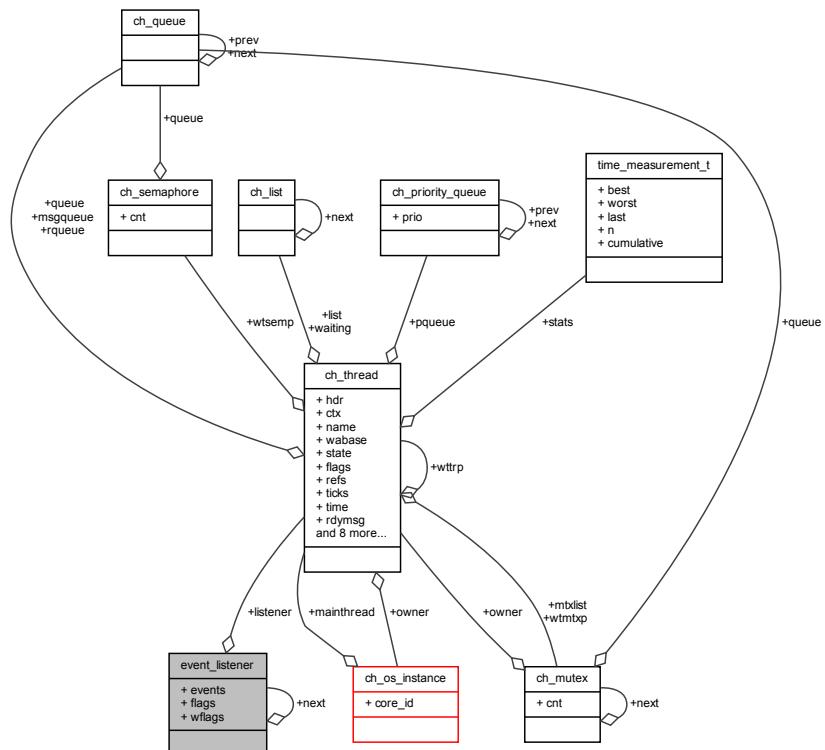
Condition variable threads queue.

8.37 event_listener Struct Reference

Event Listener structure.

```
#include <chevents.h>
```

Collaboration diagram for `event_listener`:



Data Fields

- `event_listener_t * next`
Next Event Listener registered on the event source.
- `thread_t * listener`
Thread interested in the event source.
- `eventmask_t events`
Events to be set in the listening thread.
- `eventflags_t flags`
Flags added to the listener by the event source.
- `eventflags_t wflags`
Flags that this listener interested in.

8.37.1 Detailed Description

Event Listener structure.

8.37.2 Field Documentation

8.37.2.1 `next`

`event_listener_t* event_listener::next`

Next Event Listener registered on the event source.

8.37.2.2 `listener`

`thread_t* event_listener::listener`

Thread interested in the event source.

8.37.2.3 `events`

`eventmask_t event_listener::events`

Events to be set in the listening thread.

8.37.2.4 flags

```
eventflags_t event_listener::flags
```

Flags added to the listener by the event source.

8.37.2.5 wflags

```
eventflags_t event_listener::wflags
```

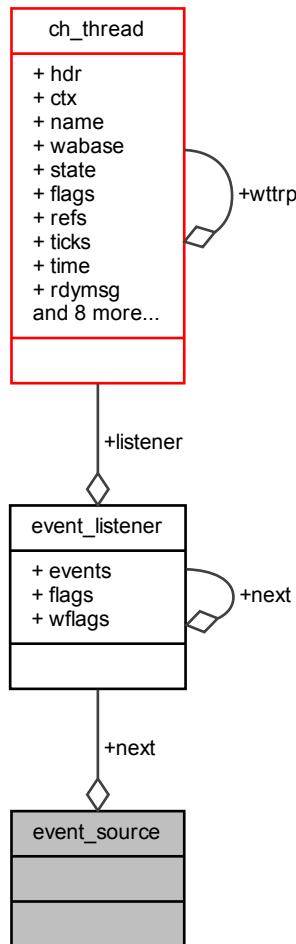
Flags that this listener interested in.

8.38 event_source Struct Reference

Event Source structure.

```
#include <chevents.h>
```

Collaboration diagram for event_source:



Data Fields

- [event_listener_t * next](#)
First Event Listener registered on the Event Source.

8.38.1 Detailed Description

Event Source structure.

8.38.2 Field Documentation

8.38.2.1 next

```
event_listener_t* event_source::next
```

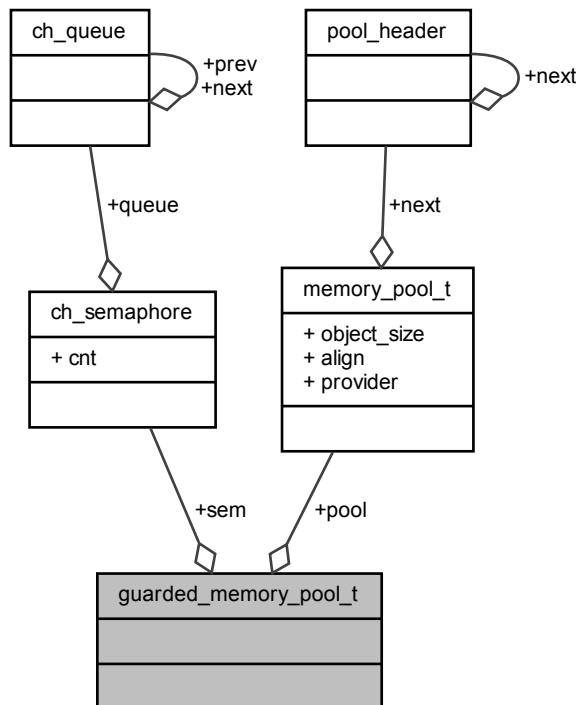
First Event Listener registered on the Event Source.

8.39 guarded_memory_pool_t Struct Reference

Guarded memory pool descriptor.

```
#include <chmempools.h>
```

Collaboration diagram for guarded_memory_pool_t:



Data Fields

- `semaphore_t sem`

Counter semaphore guarding the memory pool.

- `memory_pool_t pool`

The memory pool itself.

8.39.1 Detailed Description

Guarded memory pool descriptor.

8.39.2 Field Documentation

8.39.2.1 sem

```
semaphore_t guarded_memory_pool_t::sem
```

Counter semaphore guarding the memory pool.

8.39.2.2 pool

```
memory_pool_t guarded_memory_pool_t::pool
```

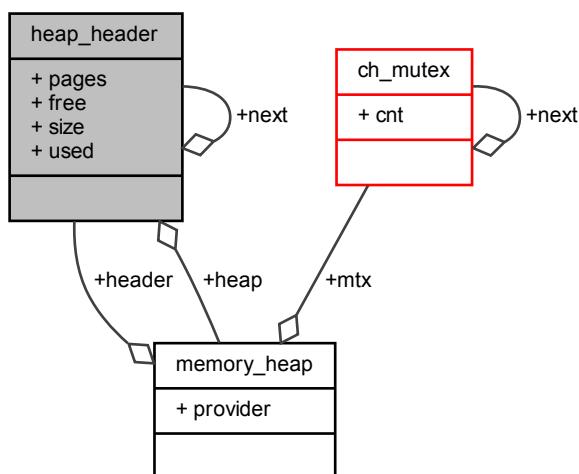
The memory pool itself.

8.40 heap_header Union Reference

Memory heap block header.

```
#include <chmemheaps.h>
```

Collaboration diagram for heap_header:



8.40.1 Detailed Description

Memory heap block header.

8.40.2 Field Documentation

8.40.2.1 next

```
heap_header_t* heap_header::next
```

Next block in free list.

8.40.2.2 pages

```
size_t heap_header::pages
```

Size of the area in pages.

8.40.2.3 heap

```
memory_heap_t* heap_header::heap
```

Block owner heap.

8.40.2.4 size

```
size_t heap_header::size
```

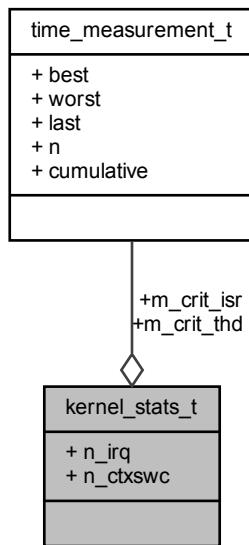
Size of the area in bytes.

8.41 kernel_stats_t Struct Reference

Type of a kernel statistics structure.

```
#include <chstats.h>
```

Collaboration diagram for kernel_stats_t:



Data Fields

- `ucnt_t n_irq`
Number of IRQs.
- `ucnt_t n_ctxswc`
Number of context switches.
- `time_measurement_t m_crit_thd`
Measurement of threads critical zones duration.
- `time_measurement_t m_crit_isr`
Measurement of ISRs critical zones duration.

8.41.1 Detailed Description

Type of a kernel statistics structure.

8.41.2 Field Documentation

8.41.2.1 n_irq

```
ucnt_t kernel_stats_t::n_irq
```

Number of IRQs.

8.41.2.2 n_ctxswc

```
ucnt_t kernel_stats_t::n_ctxswc
```

Number of context switches.

8.41.2.3 m_crit_thd

```
time_measurement_t kernel_stats_t::m_crit_thd
```

Measurement of threads critical zones duration.

8.41.2.4 m_crit_isr

```
time_measurement_t kernel_stats_t::m_crit_isr
```

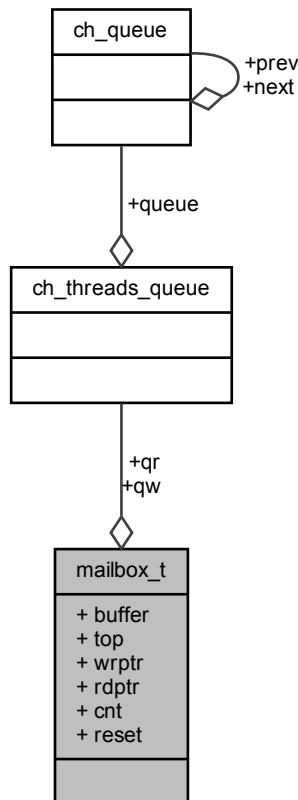
Measurement of ISRs critical zones duration.

8.42 mailbox_t Struct Reference

Structure representing a mailbox object.

```
#include <chmboxes.h>
```

Collaboration diagram for mailbox_t:



Data Fields

- `msg_t * buffer`

Pointer to the mailbox buffer.

- `msg_t * top`

Pointer to the location after the buffer.

- `msg_t * wrptr`

Write pointer.

- `msg_t * rdptr`

Read pointer.

- `size_t cnt`

Messages in queue.

- `bool reset`

True in reset state.

- `threads_queue_t qw`

Queued writers.

- `threads_queue_t qr`

Queued readers.

8.42.1 Detailed Description

Structure representing a mailbox object.

8.42.2 Field Documentation

8.42.2.1 buffer

```
msg_t* mailbox_t::buffer
```

Pointer to the mailbox buffer.

8.42.2.2 top

```
msg_t* mailbox_t::top
```

Pointer to the location after the buffer.

8.42.2.3 wrptr

```
msg_t* mailbox_t::wrptr
```

Write pointer.

8.42.2.4 rdptr

```
msg_t* mailbox_t::rdptr
```

Read pointer.

8.42.2.5 cnt

```
size_t mailbox_t::cnt
```

Messages in queue.

8.42.2.6 reset

```
bool mailbox_t::reset
```

True in reset state.

8.42.2.7 qw

```
threads_queue_t mailbox_t::qw
```

Queued writers.

8.42.2.8 qr

```
threads_queue_t mailbox_t::qr
```

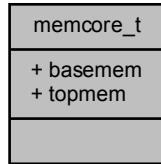
Queued readers.

8.43 memcore_t Struct Reference

Type of memory core object.

```
#include <chmemcore.h>
```

Collaboration diagram for memcore_t:



Data Fields

- `uint8_t * basemem`

Next free address.

- `uint8_t * topmem`

Final address.

8.43.1 Detailed Description

Type of memory core object.

8.43.2 Field Documentation

8.43.2.1 basemem

```
uint8_t* memcore_t::basemem
```

Next free address.

8.43.2.2 topmem

```
uint8_t* memcore_t::topmem
```

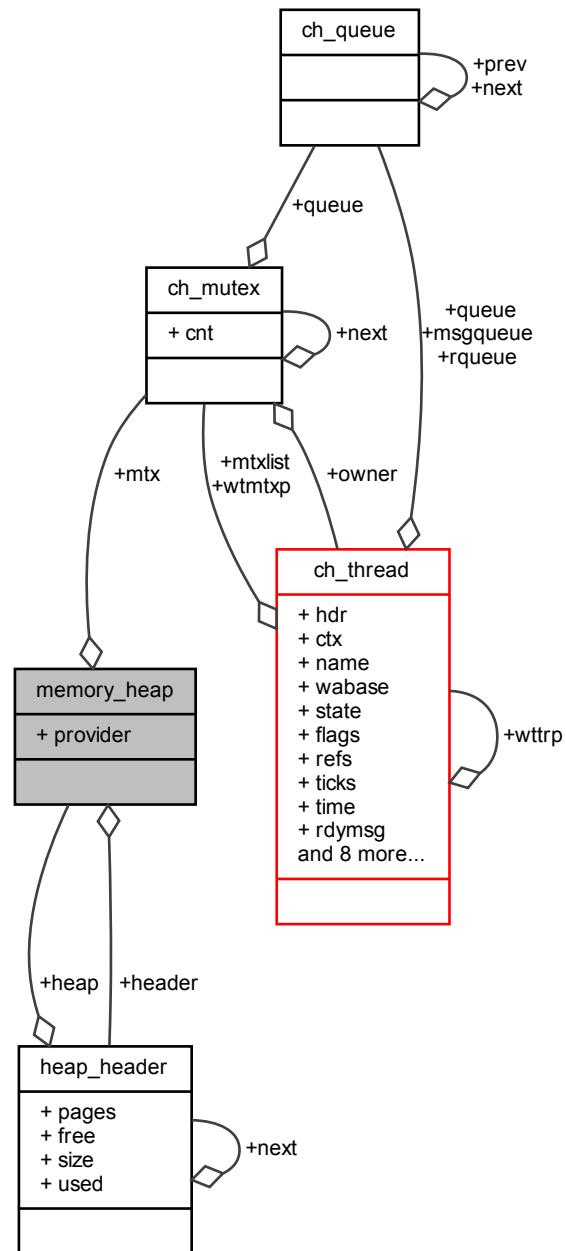
Final address.

8.44 memory_heap Struct Reference

Structure describing a memory heap.

```
#include <chmemheaps.h>
```

Collaboration diagram for memory_heap:



Data Fields

- `memgetfunc2_t provider`
Memory blocks provider for this heap.
- `heap_header_t header`
Free blocks list header.
- `mutex_t mtx`
Heap access mutex.

8.44.1 Detailed Description

Structure describing a memory heap.

8.44.2 Field Documentation

8.44.2.1 provider

`memgetfunc2_t memory_heap::provider`

Memory blocks provider for this heap.

8.44.2.2 header

`heap_header_t memory_heap::header`

Free blocks list header.

8.44.2.3 mtx

`mutex_t memory_heap::mtx`

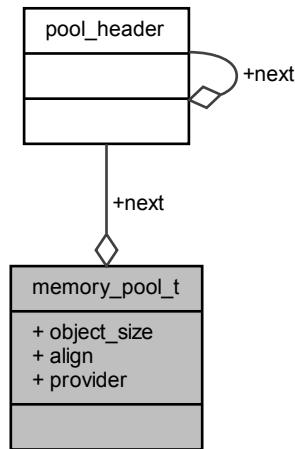
Heap access mutex.

8.45 memory_pool_t Struct Reference

Memory pool descriptor.

```
#include <chmempools.h>
```

Collaboration diagram for memory_pool_t:



Data Fields

- struct `pool_header` * `next`
Pointer to the header.
- size_t `object_size`
Memory pool objects size.
- unsigned `align`
Required alignment.
- memgetfunc_t `provider`
Memory blocks provider for this pool.

8.45.1 Detailed Description

Memory pool descriptor.

8.45.2 Field Documentation

8.45.2.1 next

```
struct pool_header* memory_pool_t::next
```

Pointer to the header.

8.45.2.2 object_size

```
size_t memory_pool_t::object_size
```

Memory pool objects size.

8.45.2.3 align

```
unsigned memory_pool_t::align
```

Required alignment.

8.45.2.4 provider

```
memgetfunc_t memory_pool_t::provider
```

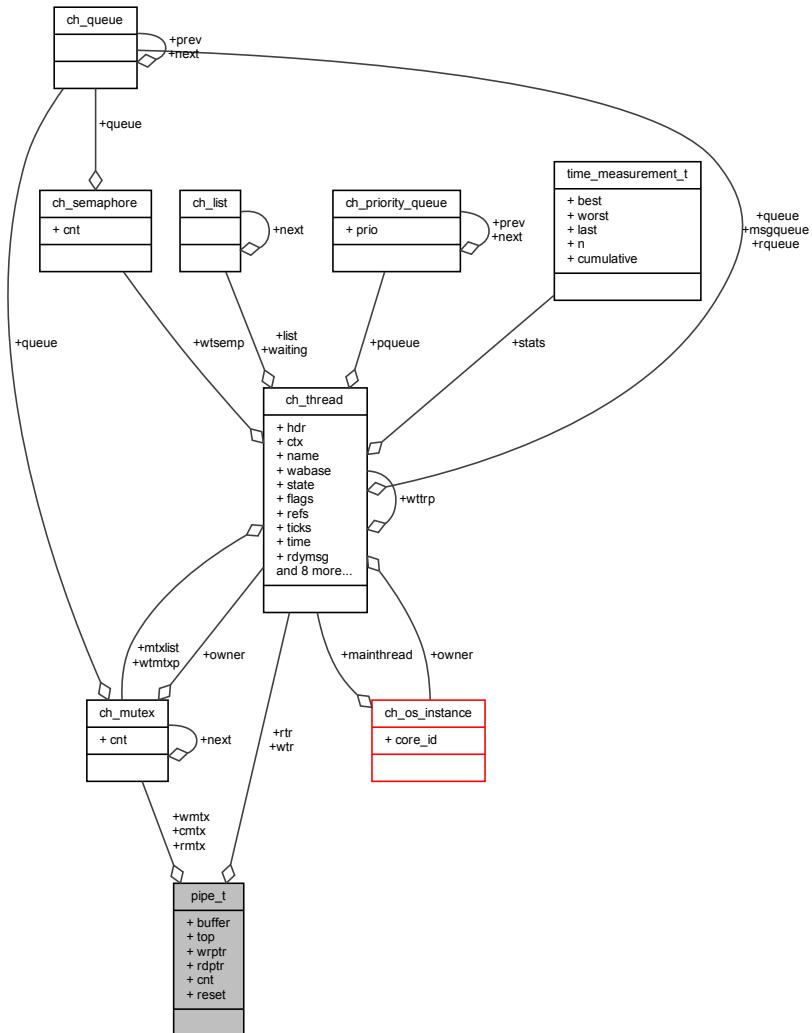
Memory blocks provider for this pool.

8.46 pipe_t Struct Reference

Structure representing a pipe object.

```
#include <chpipes.h>
```

Collaboration diagram for pipe_t:



Data Fields

- `uint8_t * buffer`

Pointer to the pipe buffer.

- `uint8_t * top`

Pointer to the location after the buffer.

- `uint8_t * wrptr`

Write pointer.

- `uint8_t * rdptr`

Read pointer.

- `size_t cnt`

Bytes in the pipe.

- `bool reset`
True if in reset state.
- `thread_reference_t wtr`
Waiting writer.
- `thread_reference_t rtr`
Waiting reader.
- `mutex_t cmtx`
Common access mutex.
- `mutex_t wmtx`
Write access mutex.
- `mutex_t rmtx`
Read access mutex.

8.46.1 Detailed Description

Structure representing a pipe object.

8.46.2 Field Documentation

8.46.2.1 buffer

```
uint8_t* pipe_t::buffer
```

Pointer to the pipe buffer.

8.46.2.2 top

```
uint8_t* pipe_t::top
```

Pointer to the location after the buffer.

8.46.2.3 wrptr

```
uint8_t* pipe_t::wrptr
```

Write pointer.

8.46.2.4 rdptr

```
uint8_t* pipe_t::rdptr
```

Read pointer.

8.46.2.5 cnt

```
size_t pipe_t::cnt
```

Bytes in the pipe.

8.46.2.6 reset

```
bool pipe_t::reset
```

True if in reset state.

8.46.2.7 wtr

```
thread_reference_t pipe_t::wtr
```

Waiting writer.

8.46.2.8 rtr

```
thread_reference_t pipe_t::rtr
```

Waiting reader.

8.46.2.9 cmtx

`mutex_t pipe_t::cmtx`

Common access mutex.

8.46.2.10 wmtx

`mutex_t pipe_t::wmtx`

Write access mutex.

8.46.2.11 rmtx

`mutex_t pipe_t::rmtx`

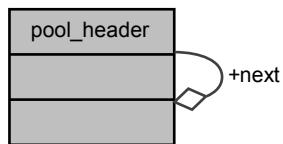
Read access mutex.

8.47 pool_header Struct Reference

Memory pool free object header.

```
#include <chmempools.h>
```

Collaboration diagram for pool_header:



Data Fields

- struct `pool_header` * `next`

Pointer to the next pool header in the list.

8.47.1 Detailed Description

Memory pool free object header.

8.47.2 Field Documentation

8.47.2.1 next

```
struct pool_header* pool_header::next
```

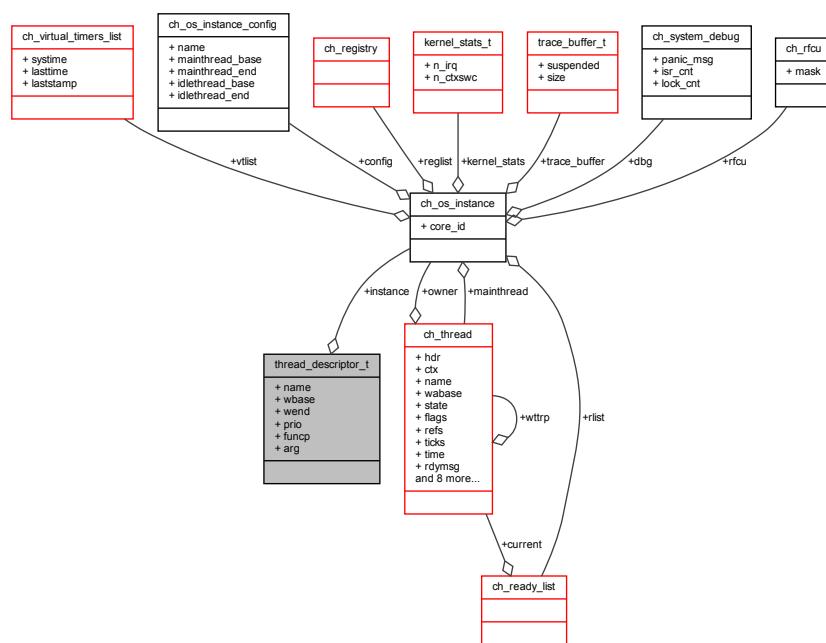
Pointer to the next pool header in the list.

8.48 thread_descriptor_t Struct Reference

Type of a thread descriptor.

```
#include <chthreads.h>
```

Collaboration diagram for thread_descriptor_t:



Data Fields

- `const char * name`
Thread name.
- `stkalign_t * wbase`
Pointer to the working area base.
- `stkalign_t * wend`
Pointer to the working area end.
- `tprio_t prio`
Thread priority.
- `tfunc_t funcp`
Thread function pointer.
- `void * arg`
Thread argument.
- `os_instance_t * instance`
OS instance affinity or `NULL` for current one.

8.48.1 Detailed Description

Type of a thread descriptor.

8.48.2 Field Documentation

8.48.2.1 name

```
const char* thread_descriptor_t::name
```

Thread name.

8.48.2.2 wbase

```
stkalign_t* thread_descriptor_t::wbase
```

Pointer to the working area base.

8.48.2.3 wend

```
stkalign_t* thread_descriptor_t::wend
```

Pointer to the working area end.

8.48.2.4 prio

```
tprio_t thread_descriptor_t::prio
```

Thread priority.

8.48.2.5 funcp

```
tfunc_t thread_descriptor_t::funcp
```

Thread function pointer.

8.48.2.6 arg

```
void* thread_descriptor_t::arg
```

Thread argument.

8.48.2.7 instance

```
os_instance_t* thread_descriptor_t::instance
```

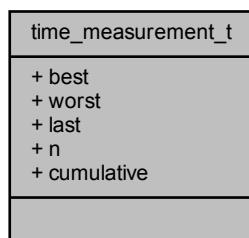
OS instance affinity or NULL for current one.

8.49 time_measurement_t Struct Reference

Type of a Time Measurement object.

```
#include <chtm.h>
```

Collaboration diagram for time_measurement_t:



Data Fields

- `rtcnt_t best`
Best measurement.
- `rtcnt_t worst`
Worst measurement.
- `rtcnt_t last`
Last measurement.
- `ucnt_t n`
Number of measurements.
- `rttime_t cumulative`
Cumulative measurement.

8.49.1 Detailed Description

Type of a Time Measurement object.

Note

The maximum measurable time period depends on the implementation of the realtime counter and its clock frequency.

The measurement is not 100% cycle-accurate, it can be in excess of few cycles depending on the compiler and target architecture.

Interrupts can affect measurement if the measurement is performed with interrupts enabled.

8.49.2 Field Documentation

8.49.2.1 best

```
rtcnt_t time_measurement_t::best
```

Best measurement.

8.49.2.2 worst

```
rtcnt_t time_measurement_t::worst
```

Worst measurement.

8.49.2.3 last

```
rtcnt_t time_measurement_t::last
```

Last measurement.

8.49.2.4 n

```
ucnt_t time_measurement_t::n
```

Number of measurements.

8.49.2.5 cumulative

```
rttime_t time_measurement_t::cumulative
```

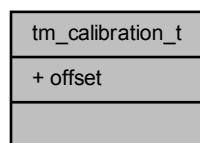
Cumulative measurement.

8.50 tm_calibration_t Struct Reference

Type of a time measurement calibration data.

```
#include <chtm.h>
```

Collaboration diagram for tm_calibration_t:



Data Fields

- `rtcnt_t offset`

Measurement calibration value.

8.50.1 Detailed Description

Type of a time measurement calibration data.

8.50.2 Field Documentation

8.50.2.1 offset

`rtcnt_t tm_calibration_t::offset`

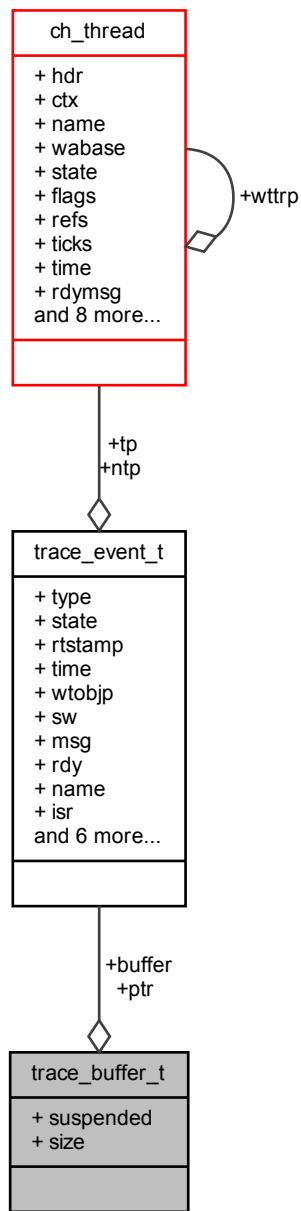
Measurement calibration value.

8.51 trace_buffer_t Struct Reference

Trace buffer header.

```
#include <chtrace.h>
```

Collaboration diagram for trace_buffer_t:



Data Fields

- `uint16_t suspended`
Suspended trace sources mask.
- `uint16_t size`
Trace buffer size (entries).
- `trace_event_t * ptr`
Pointer to the buffer front.

- `trace_event_t buffer [CH_DBG_TRACE_BUFFER_SIZE]`
Ring buffer.

8.51.1 Detailed Description

Trace buffer header.

8.51.2 Field Documentation

8.51.2.1 suspended

```
uint16_t trace_buffer_t::suspended
```

Suspended trace sources mask.

8.51.2.2 size

```
uint16_t trace_buffer_t::size
```

Trace buffer size (entries).

8.51.2.3 ptr

```
trace_event_t* trace_buffer_t::ptr
```

Pointer to the buffer front.

8.51.2.4 buffer

```
trace_event_t trace_buffer_t::buffer[CH_DBG_TRACE_BUFFER_SIZE]
```

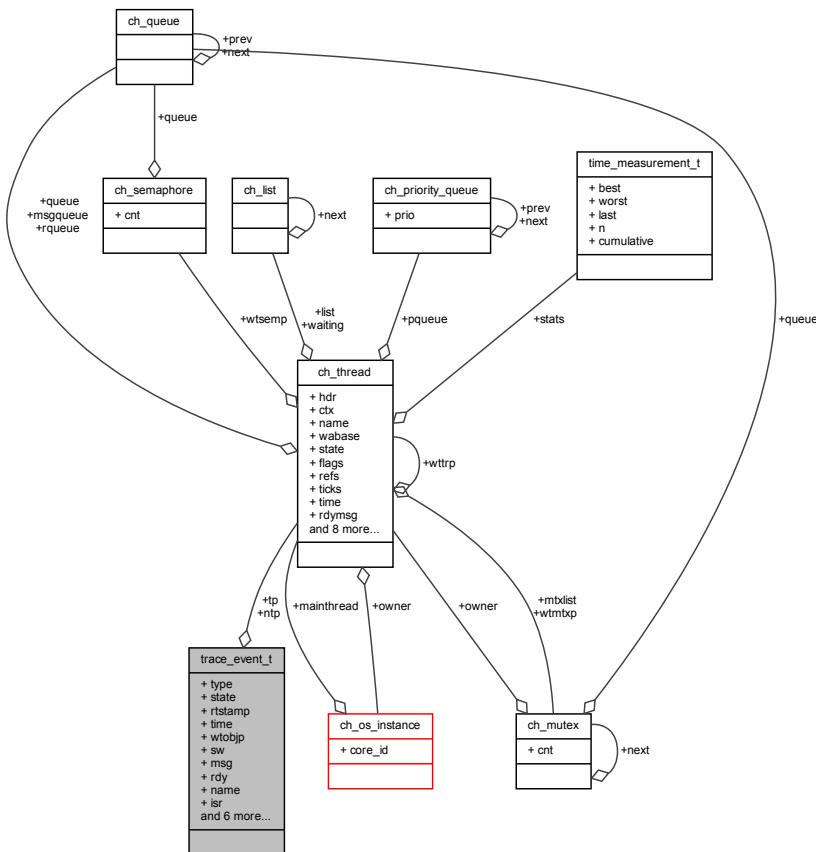
Ring buffer.

8.52 trace_event_t Struct Reference

Trace buffer record.

```
#include <chtrace.h>
```

Collaboration diagram for trace_event_t:



Data Fields

- `uint32_t type:3`
Record type.
- `uint32_t state:5`
Switched out thread state.
- `uint32_t rtimestamp:24`
Accurate time stamp.
- `systime_t time`
System time stamp of the switch event.
- `thread_t * ntp`
Switched in thread.
- `void * wtobjp`
Object where going to sleep.

- struct {

 thread_t * ntp

 Switched in thread.

 void * wtobjp

 Object where going to sleep.

} sw

 Structure representing a context switch.
- thread_t * tp

 Thread made ready.
- msg_t msg

 Ready message.
- struct {

 thread_t * tp

 Thread made ready.

 msg_t msg

 Ready message.

} rdy

 Structure representing a thread becoming ready.
- const char * name

 *ISR function name taken using **func**.*
- struct {

 const char * name

 *ISR function name taken using **func**.*

} isr

 Structure representing an ISR enter.
- const char * reason

 Halt error string.
- struct {

 const char * reason

 Halt error string.

} halt

 Structure representing an halt.
- void * up1

 Trace user parameter 1.
- void * up2

 Trace user parameter 2.
- struct {

 void * up1

 Trace user parameter 1.

 void * up2

 Trace user parameter 2.

} user

 User trace structure.

8.52.1 Detailed Description

Trace buffer record.

8.52.2 Field Documentation

8.52.2.1 type

```
uint32_t trace_event_t::type
```

Record type.

8.52.2.2 state

```
uint32_t trace_event_t::state
```

Switched out thread state.

8.52.2.3 rtstamp

```
uint32_t trace_event_t::rtstamp
```

Accurate time stamp.

Note

This field only available if the port supports PORT_SUPPORTS_RT else it is set to zero.

8.52.2.4 time

```
sysime_t trace_event_t::time
```

System time stamp of the switch event.

8.52.2.5 ntp

```
thread_t* trace_event_t::ntp
```

Switched in thread.

8.52.2.6 wtobjp

```
void* trace_event_t::wtobjp
```

Object where going to sleep.

8.52.2.7 sw

```
struct { ... } trace_event_t::sw
```

Structure representing a context switch.

8.52.2.8 tp

```
thread_t* trace_event_t::tp
```

Thread made ready.

8.52.2.9 msg

```
msg_t trace_event_t::msg
```

Ready message.

8.52.2.10 rdy

```
struct { ... } trace_event_t::rdy
```

Structure representing a thread becoming ready.

8.52.2.11 name

```
const char* trace_event_t::name
```

ISR function name taken using **func**.

8.52.2.12 isr

```
struct { ... } trace_event_t::isr
```

Structure representing an ISR enter.

8.52.2.13 reason

```
const char* trace_event_t::reason
```

Halt error string.

8.52.2.14 halt

```
struct { ... } trace_event_t::halt
```

Structure representing an halt.

8.52.2.15 up1

```
void* trace_event_t::up1
```

Trace user parameter 1.

8.52.2.16 up2

```
void* trace_event_t::up2
```

Trace user parameter 2.

8.52.2.17 user

```
struct { ... } trace_event_t::user
```

User trace structure.

Chapter 9

File Documentation

9.1 ch.h File Reference

ChibiOS/RT main include file.

```
#include "chlicense.h"
#include "chconf.h"
#include "chchecks.h"
#include "chrestrictions.h"
#include "clearly.h"
#include "chrfcu.h"
#include "chdebug.h"
#include "ctime.h"
#include "chlists.h"
#include "chalign.h"
#include "chtrace.h"
#include "chport.h"
#include "chtm.h"
#include "chstats.h"
#include "chobjects.h"
#include "chsys.h"
#include "chinstances.h"
#include "chvt.h"
#include "chsched.h"
#include "chthreads.h"
#include "chregistry.h"
#include "chsem.h"
#include "chmtx.h"
#include "chcond.h"
#include "chevents.h"
#include "chmsg.h"
#include "chlib.h"
#include "chdynamic.h"
```

Macros

- `#define __CHIBIOS_RT__`

ChibiOS/RT identification macro.

- `#define CH_KERNEL_STABLE 0`
Stable release flag.

ChibiOS/RT version identification

- `#define CH_KERNEL_VERSION "7.0.0"`
Kernel version string.
- `#define CH_KERNEL_MAJOR 7`
Kernel version major number.
- `#define CH_KERNEL_MINOR 0`
Kernel version minor number.
- `#define CH_KERNEL_PATCH 0`
Kernel version patch number.

Constants for configuration options

- `#define FALSE 0`
Generic 'false' preprocessor boolean constant.
- `#define TRUE 1`
Generic 'true' preprocessor boolean constant.

9.1.1 Detailed Description

ChibiOS/RT main include file.

9.2 chalign.h File Reference

Memory alignment macros and structures.

Macros

Memory alignment support macros

- `#define MEM_ALIGN_MASK(a) ((size_t)(a) - 1U)`
Alignment mask constant.
- `#define MEM_ALIGN_PREV(p, a)`
Aligns to the previous aligned memory address.
- `#define MEM_ALIGN_NEXT(p, a)`
Aligns to the next aligned memory address.
- `#define MEM_IS_ALIGNED(p, a) (((size_t)(p) & MEM_ALIGN_MASK(a)) == 0U)`
Returns whatever a pointer or memory size is aligned.
- `#define MEM_IS_VALID_ALIGNMENT(a) (((size_t)(a) != 0U) && (((size_t)(a) & ((size_t)(a) - 1U)) == 0U))`
Returns whatever a constant is a valid alignment.

9.2.1 Detailed Description

Memory alignment macros and structures.

9.3 chbsem.h File Reference

Binary semaphores structures and macros.

Data Structures

- struct `ch_binary_semaphore`

Binary semaphore type.

Macros

- `#define __BSEMAPHORE_DATA(name, taken) {__SEMAPHORE_DATA(name.sem, ((taken) ? 0 : 1))}`
Data part of a static semaphore initializer.
- `#define BSEMAPHORE_DECL(name, taken) binary_semaphore_t name = __BSEMAPHORE_DATA(name, taken)`
Static semaphore initializer.

Typedefs

- `typedef struct ch_binary_semaphore binary_semaphore_t`
Binary semaphore type.

Functions

- `static void chBSemObjectInit (binary_semaphore_t *bsp, bool taken)`
Initializes a binary semaphore.
- `static msg_t chBSemWait (binary_semaphore_t *bsp)`
Wait operation on the binary semaphore.
- `static msg_t chBSemWaitS (binary_semaphore_t *bsp)`
Wait operation on the binary semaphore.
- `static msg_t chBSemWaitTimeoutS (binary_semaphore_t *bsp, sysinterval_t timeout)`
Wait operation on the binary semaphore.
- `static msg_t chBSemWaitTimeout (binary_semaphore_t *bsp, sysinterval_t timeout)`
Wait operation on the binary semaphore.
- `static void chBSemResetl (binary_semaphore_t *bsp, bool taken)`
Reset operation on the binary semaphore.
- `static void chBSemReset (binary_semaphore_t *bsp, bool taken)`
Reset operation on the binary semaphore.
- `static void chBSemSignall (binary_semaphore_t *bsp)`
Performs a signal operation on a binary semaphore.
- `static void chBSemSignal (binary_semaphore_t *bsp)`
Performs a signal operation on a binary semaphore.
- `static bool chBSemGetStatel (const binary_semaphore_t *bsp)`
Returns the binary semaphore current state.

9.3.1 Detailed Description

Binary semaphores structures and macros.

Binary semaphores related APIs and services.

Operation mode

Binary semaphores are implemented as a set of inline functions that use the existing counting semaphores primitives. The difference between counting and binary semaphores is that the counter of binary semaphores is not allowed to grow above the value 1. Repeated signal operation are ignored. A binary semaphore can thus have only two defined states:

- **Taken**, when its counter has a value of zero or lower than zero. A negative number represent the number of threads queued on the binary semaphore.
- **Not taken**, when its counter has a value of one.

Binary semaphores are different from mutexes because there is no concept of ownership, a binary semaphore can be taken by a thread and signaled by another thread or an interrupt handler, mutexes can only be taken and released by the same thread. Another difference is that binary semaphores, unlike mutexes, do not implement the priority inheritance protocol.

In order to use the binary semaphores APIs the `CH_CFG_USE_SEMAPHORES` option must be enabled in `chconf.h`.

9.4 chchecks.h File Reference

Configuration file checks header.

9.4.1 Detailed Description

Configuration file checks header.

9.5 chcond.c File Reference

Condition Variables code.

```
#include "ch.h"
```

Functions

- void `chCondObjectInit` (`condition_variable_t` *cp)
Initializes a condition_variable_t structure.
- void `chCondSignal` (`condition_variable_t` *cp)
Signals one thread that is waiting on the condition variable.
- void `chCondSignall` (`condition_variable_t` *cp)
Signals one thread that is waiting on the condition variable.
- void `chCondBroadcast` (`condition_variable_t` *cp)
Signals all threads that are waiting on the condition variable.
- void `chCondBroadcastl` (`condition_variable_t` *cp)
Signals all threads that are waiting on the condition variable.
- `msg_t chCondWait` (`condition_variable_t` *cp)
Waits on the condition variable releasing the mutex lock.
- `msg_t chCondWaitS` (`condition_variable_t` *cp)
Waits on the condition variable releasing the mutex lock.
- `msg_t chCondWaitTimeout` (`condition_variable_t` *cp, `sysinterval_t` timeout)
Waits on the condition variable releasing the mutex lock.
- `msg_t chCondWaitTimeoutS` (`condition_variable_t` *cp, `sysinterval_t` timeout)
Waits on the condition variable releasing the mutex lock.

9.5.1 Detailed Description

Condition Variables code.

9.6 chcond.h File Reference

Condition Variables macros and structures.

Data Structures

- struct `condition_variable`
condition_variable_t structure.

Macros

- `#define __CONDVAR_DATA(name) {__CH_QUEUE_DATA(name.queue)}`
Data part of a static condition variable initializer.
- `#define CONDVAR_DECL(name) condition_variable_t name = __CONDVAR_DATA(name)`
Static condition variable initializer.

Typedefs

- `typedef struct condition_variable condition_variable_t`
condition_variable_t structure.

Functions

- void `chCondObjectInit` (`condition_variable_t` *cp)
Initializes a condition_variable_t structure.
- void `chCondSignal` (`condition_variable_t` *cp)
Signals one thread that is waiting on the condition variable.
- void `chCondSignall` (`condition_variable_t` *cp)
Signals one thread that is waiting on the condition variable.
- void `chCondBroadcast` (`condition_variable_t` *cp)
Signals all threads that are waiting on the condition variable.
- void `chCondBroadcastl` (`condition_variable_t` *cp)
Signals all threads that are waiting on the condition variable.
- `msg_t chCondWait` (`condition_variable_t` *cp)
Waits on the condition variable releasing the mutex lock.
- `msg_t chCondWaitS` (`condition_variable_t` *cp)
Waits on the condition variable releasing the mutex lock.
- `msg_t chCondWaitTimeout` (`condition_variable_t` *cp, `sysinterval_t` timeout)
Waits on the condition variable releasing the mutex lock.
- `msg_t chCondWaitTimeoutS` (`condition_variable_t` *cp, `sysinterval_t` timeout)
Waits on the condition variable releasing the mutex lock.

9.6.1 Detailed Description

Condition Variables macros and structures.

9.7 chconf.h File Reference

Configuration file template.

Macros

System settings

- #define `CH_CFG_SMP_MODE FALSE`
Handling of instances.

System timers settings

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

Kernel parameters and options

- #define CH_CFG_TIME_QUANTUM 0
Round robin interval.
- #define CH_CFG_NO_IDLE_THREAD FALSE
Idle thread automatic spawn suppression.

Performance options

- #define CH_CFG_OPTIMIZE_SPEED TRUE
OS optimization.

Subsystem options

- #define CH_CFG_USE_TM TRUE
Time Measurement APIs.
- #define CH_CFG_USE_TIMESTAMP TRUE
Time Stamps APIs.
- #define CH_CFG_USE_REGISTRY TRUE
Threads registry APIs.
- #define CH_CFG_USE_WAITEXIT TRUE
Threads synchronization APIs.
- #define CH_CFG_USE_SEMAPHORES TRUE
Semaphores APIs.
- #define CH_CFG_USE_SEMAPHORES_PRIORITY FALSE
Semaphores queuing mode.
- #define CH_CFG_USE_MUTEXES TRUE
Mutexes APIs.
- #define CH_CFG_USE_MUTEXES_RECURSIVE FALSE
Enables recursive behavior on mutexes.
- #define CH_CFG_USE_CONDVARNS TRUE
Conditional Variables APIs.
- #define CH_CFG_USE_CONDVARNS_TIMEOUT TRUE
Conditional Variables APIs with timeout.
- #define CH_CFG_USE_EVENTS TRUE
Events Flags APIs.
- #define CH_CFG_USE_EVENTS_TIMEOUT TRUE
Events Flags APIs with timeout.
- #define CH_CFG_USE_MESSAGES TRUE
Synchronous Messages APIs.
- #define CH_CFG_USE_MESSAGES_PRIORITY FALSE
Synchronous Messages queuing mode.
- #define CH_CFG_USE_DYNAMIC TRUE
Dynamic Threads APIs.

OSLIB options

- #define CH_CFG_USE_MAILBOXES TRUE
Mailboxes APIs.
- #define CH_CFG_USE_MEMCORE TRUE
Core Memory Manager APIs.
- #define CH_CFG_MEMCORE_SIZE 0
Managed RAM size.
- #define CH_CFG_USE_HEAP TRUE
Heap Allocator APIs.
- #define CH_CFG_USE_MEMPOOLS TRUE
Memory Pools Allocator APIs.

- #define CH_CFG_USE_OBJ_FIFOS TRUE
Objects FIFOs APIs.
- #define CH_CFG_USE_PIPES TRUE
Pipes APIs.
- #define CH_CFG_USE_OBJ_CACHES TRUE
Objects Caches APIs.
- #define CH_CFG_USE_DELEGATES TRUE
Delegate threads APIs.
- #define CH_CFG_USE_JOBS TRUE
Jobs Queues APIs.

Objects factory options

- #define CH_CFG_USE_FACTORY TRUE
Objects Factory APIs.
- #define CH_CFG_FACTORY_MAX_NAMES_LENGTH 8
Maximum length for object names.
- #define CH_CFG_FACTORY_OBJECTS_REGISTRY TRUE
Enables the registry of generic objects.
- #define CH_CFG_FACTORY_GENERIC_BUFFERS TRUE
Enables factory for generic buffers.
- #define CH_CFG_FACTORY_SEMAPHORES TRUE
Enables factory for semaphores.
- #define CH_CFG_FACTORY_MAILBOXES TRUE
Enables factory for mailboxes.
- #define CH_CFG_FACTORY_OBJ_FIFOS TRUE
Enables factory for objects FIFOs.
- #define CH_CFG_FACTORY_PIPES TRUE
Enables factory for Pipes.

Debug options

- #define CH_DBG_STATISTICS FALSE
Debug option, kernel statistics.
- #define CH_DBG_SYSTEM_STATE_CHECK TRUE
Debug option, system state check.
- #define CH_DBG_ENABLE_CHECKS TRUE
Debug option, parameters checks.
- #define CH_DBG_ENABLE_ASSERTS TRUE
Debug option, consistency checks.
- #define CH_DBG_TRACE_MASK CH_DBG_TRACE_MASK_ALL
Debug option, trace buffer.
- #define CH_DBG_TRACE_BUFFER_SIZE 128
Trace buffer entries.
- #define CH_DBG_ENABLE_STACK_CHECK TRUE
Debug option, stack checks.
- #define CH_DBG_FILL_THREADS TRUE
Debug option, stacks initialization.
- #define CH_DBG_THREADS_PROFILING FALSE
Debug option, threads profiling.

Kernel hooks

- #define CH_CFG_SYSTEM_EXTRA_FIELDS /* Add system custom fields here.*/
System structure extension.
- #define CH_CFG_SYSTEM_INIT_HOOK()
System initialization hook.

- #define CH_CFG_OS_INSTANCE_EXTRA_FIELDS /* Add OS instance custom fields here.*/
OS instance structure extension.
- #define CH_CFG_OS_INSTANCE_INIT_HOOK(oip)
OS instance initialization hook.
- #define CH_CFG_THREAD_EXTRA_FIELDS /* Add threads custom fields here.*/
Threads descriptor structure extension.
- #define CH_CFG_THREAD_INIT_HOOK(tp)
Threads initialization hook.
- #define CH_CFG_THREAD_EXIT_HOOK(tp)
Threads finalization hook.
- #define CH_CFG_CONTEXT_SWITCH_HOOK(ntp, otp)
Context switch hook.
- #define CH_CFG_IRQ_PROLOGUE_HOOK()
ISR enter hook.
- #define CH_CFG_IRQ_EPILOGUE_HOOK()
ISR exit hook.
- #define CH_CFG_IDLE_ENTER_HOOK()
Idle thread enter hook.
- #define CH_CFG_IDLE_LEAVE_HOOK()
Idle thread leave hook.
- #define CH_CFG_IDLE_LOOP_HOOK()
Idle Loop hook.
- #define CH_CFG_SYSTEM_TICK_HOOK()
System tick event hook.
- #define CH_CFG_SYSTEM_HALT_HOOK(reason)
System halt hook.
- #define CH_CFG_TRACE_HOOK(tep)
Trace hook.
- #define CH_CFG_RUNTIME_FAULTS_HOOK(mask)
Runtime Faults Collection Unit hook.

9.7.1 Detailed Description

Configuration file template.

A copy of this file must be placed in each project directory, it contains the application specific kernel settings.

9.8 chcustomer.h File Reference

Customer-related info.

Macros

- #define CH_CUSTOMER_ID_STRING "Santa, North Pole"
Customer readable identifier.
- #define CH_CUSTOMER_ID_CODE "xxxx-yyyy"
Customer code.
- #define CH_CUSTOMER_LICENSE_EOS_DATE 209912
End-Of-Support date (yyyymm).
- #define CH_CUSTOMER_LICENSE_VERSION_YEAR 99
Licensed branch year.

- #define CH_CUSTOMER_LICENSE_VERSION_MONTH 12
Licensed branch month.
- #define CH_LICENSE CH_LICENSE_GPL
Current license.
- #define CH_CUSTOMER_LICENSE_VERSION_DATE
Licensed version date in numeric form (yyyymm).

Licensed Products

- #define CH_CUSTOMER_LIC_RT TRUE
- #define CH_CUSTOMER_LIC_NIL TRUE
- #define CH_CUSTOMER_LIC_OSLIB TRUE
- #define CH_CUSTOMER_LIC_EX TRUE
- #define CH_CUSTOMER_LIC_SB TRUE
- #define CH_CUSTOMER_LIC_PORT_CM0 TRUE
- #define CH_CUSTOMER_LIC_PORT_CM3 TRUE
- #define CH_CUSTOMER_LIC_PORT_CM4 TRUE
- #define CH_CUSTOMER_LIC_PORT_CM7 TRUE
- #define CH_CUSTOMER_LIC_PORT_CM33 TRUE
- #define CH_CUSTOMER_LIC_PORT_ARM79 TRUE
- #define CH_CUSTOMER_LIC_PORT_E200Z0 TRUE
- #define CH_CUSTOMER_LIC_PORT_E200Z2 TRUE
- #define CH_CUSTOMER_LIC_PORT_E200Z3 TRUE
- #define CH_CUSTOMER_LIC_PORT_E200Z4 TRUE

9.8.1 Detailed Description

Customer-related info.

9.9 chdebug.c File Reference

Debug support 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)

- void `__dbg_check_enter_isr` (void)
Guard code for chSysUnlockFromIsr () .
- void `__dbg_check_leave_isr` (void)
Guard code for CH IRQ PROLOGUE () .
- void `chDbgCheckClassI` (void)
I-class functions context check.
- void `chDbgCheckClassS` (void)
S-class functions context check.

9.9.1 Detailed Description

Debug support code.

9.10 chdebug.h File Reference

Debug support macros and structures.

Data Structures

- struct `ch_system_debug`
System debug data structure.

Macros

Debug related settings

- #define `CH_DBG_STACK_FILL_VALUE` 0x55
Fill value for thread stack area in debug mode.

Macro Functions

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

Typedefs

- typedef struct `ch_system_debug system_debug_t`
System debug data structure.

Functions

- static void `__dbg_object_init (system_debug_t *sdp)`
Debug support initialization.

9.10.1 Detailed Description

Debug support macros and structures.

9.11 chdelegates.c File Reference

Delegate threads code.

```
#include "ch.h"
```

Functions

- [`msg_t __ch_delegate_fn0 \(va_list *argsp\)`](#)
Veneer for functions with no parameters.
- [`msg_t __ch_delegate_fn1 \(va_list *argsp\)`](#)
Veneer for functions with one parameter.
- [`msg_t __ch_delegate_fn2 \(va_list *argsp\)`](#)
Veneer for functions with two parameters.
- [`msg_t __ch_delegate_fn3 \(va_list *argsp\)`](#)
Veneer for functions with three parameters.
- [`msg_t __ch_delegate_fn4 \(va_list *argsp\)`](#)
Veneer for functions with four parameters.
- [`msg_t chDelegateCallVeneer \(thread_t *tp, delegate_veneer_t veneer,...\)`](#)
Triggers a function call on a delegate thread.
- [`void chDelegateDispatch \(void\)`](#)
Call messages dispatching.
- [`msg_t chDelegateDispatchTimeout \(sysinterval_t timeout\)`](#)
Call messages dispatching with timeout.

9.11.1 Detailed Description

Delegate threads code.

Delegate threads.

Operation mode

A delegate thread is a thread performing function calls triggered by other threads. This functionality is especially useful when encapsulating a library not designed for threading into a delegate thread. Other threads have access to the library without having to worry about mutual exclusion.

Precondition

In order to use the pipes APIs the `CH_CFG_USE_DELEGATES` option must be enabled in `chconf.h`.

Note

Compatible with RT and NIL.

9.12 chdelegates.h File Reference

Delegate threads macros and structures.

```
#include <stdarg.h>
```

Typedefs

- **typedef msg_t(* delegate_veneer_t) (va_list *argsp)**
Type of a delegate veneer function.
- **typedef msg_t(* delegate_fn0_t) (void)**
Type of a delegate function with no parameters.
- **typedef msg_t(* delegate_fn1_t) (msg_t p1)**
Type of a delegate function with one parameter.
- **typedef msg_t(* delegate_fn2_t) (msg_t p1, msg_t p2)**
Type of a delegate function with two parameters.
- **typedef msg_t(* delegate_fn3_t) (msg_t p1, msg_t p2, msg_t p3)**
Type of a delegate function with three parameters.
- **typedef msg_t(* delegate_fn4_t) (msg_t p1, msg_t p2, msg_t p3, msg_t p4)**
Type of a delegate function with four parameters.

Functions

- **msg_t __ch_delegate_fn0 (va_list *argsp)**
Veneer for functions with no parameters.
- **msg_t __ch_delegate_fn1 (va_list *argsp)**
Veneer for functions with one parameter.
- **msg_t __ch_delegate_fn2 (va_list *argsp)**
Veneer for functions with two parameters.
- **msg_t __ch_delegate_fn3 (va_list *argsp)**
Veneer for functions with three parameters.
- **msg_t __ch_delegate_fn4 (va_list *argsp)**
Veneer for functions with four parameters.
- **void chDelegateDispatch (void)**
Call messages dispatching.
- **msg_t chDelegateDispatchTimeout (sysinterval_t timeout)**
Call messages dispatching with timeout.
- **msg_t chDelegateCallVeneer (thread_t *tp, delegate_veneer_t veneer,...)**
Triggers a function call on a delegate thread.
- **static msg_t chDelegateCallDirect0 (thread_t *tp, delegate_fn0_t func)**
Direct call to a function with no parameters.
- **static msg_t chDelegateCallDirect1 (thread_t *tp, delegate_fn1_t func, msg_t p1)**
Direct call to a function with one parameter.
- **static msg_t chDelegateCallDirect2 (thread_t *tp, delegate_fn2_t func, msg_t p1, msg_t p2)**
Direct call to a function with two parameters.
- **static msg_t chDelegateCallDirect3 (thread_t *tp, delegate_fn3_t func, msg_t p1, msg_t p2, msg_t p3)**
Direct call to a function with three parameters.
- **static msg_t chDelegateCallDirect4 (thread_t *tp, delegate_fn4_t func, msg_t p1, msg_t p2, msg_t p3, msg_t p4)**
Direct call to a function with four parameters.

9.12.1 Detailed Description

Delegate threads macros and structures.

9.13 chdynamic.c File Reference

Dynamic threads code.

```
#include "ch.h"
```

Functions

- `thread_t * chThdCreateFromHeap (memory_heap_t *heapp, size_t size, const char *name, tprio_t prio, tfunc_t pf, void *arg)`
Creates a new thread allocating the memory from the heap.
- `thread_t * chThdCreateFromMemoryPool (memory_pool_t *mp, const char *name, tprio_t prio, tfunc_t pf, void *arg)`
Creates a new thread allocating the memory from the specified memory pool.

9.13.1 Detailed Description

Dynamic threads code.

9.14 chdynamic.h File Reference

Dynamic threads macros and structures.

Functions

- `thread_t * chThdCreateFromHeap (memory_heap_t *heapp, size_t size, const char *name, tprio_t prio, tfunc_t pf, void *arg)`
Creates a new thread allocating the memory from the heap.
- `thread_t * chThdCreateFromMemoryPool (memory_pool_t *mp, const char *name, tprio_t prio, tfunc_t pf, void *arg)`
Creates a new thread allocating the memory from the specified memory pool.

9.14.1 Detailed Description

Dynamic threads macros and structures.

9.15 clearly.h File Reference

Early forward types declarations header.

```
#include "chtypes.h"
```

Macros

- `#define __CH_STRINGIFY(a) #a`
Utility to make the parameter a quoted string.
- `#define __CH_OFFSETOF(st, m)`
Structure field offset utility.
- `#define __CH_USED(x) (void)(x)`
Marks an expression result as used.
- `#define likely(x) PORT_LIKELY(x)`
Marks a boolean expression as likely true.
- `#define unlikely(x) PORT_UNLIKELY(x)`
Marks a boolean expression as likely false.

Typedefs

- `typedef unsigned core_id_t`
Type of a core identifier.
- `typedef struct ch_thread thread_t`
Type of a thread structure.
- `typedef struct ch_os_instance os_instance_t`
Type of an OS instance structure.

Kernel types

- `typedef port_rtcnt_t rtcnt_t`
- `typedef port_rtttime_t rtttime_t`
- `typedef port_syssts_t syssts_t`
- `typedef port_stkalign_t stkalign_t`
- `typedef uint8_t tmode_t`
- `typedef uint8_t tstate_t`
- `typedef uint8_t trefs_t`
- `typedef uint8_t tslices_t`
- `typedef uint32_t tprio_t`
- `typedef int32_t msg_t`
- `typedef int32_t eventid_t`
- `typedef uint32_t eventmask_t`
- `typedef uint32_t eventflags_t`
- `typedef int32_t cnt_t`
- `typedef uint32_t ucnt_t`

Functions

- `void chSysHalt (const char *reason)`
Halts the system.

9.15.1 Detailed Description

Early forward types declarations header.

9.16 chevents.c File Reference

Events code.

```
#include "ch.h"
```

Functions

- void `chEvtRegisterMaskWithFlagsI` (`event_source_t` *esp, `event_listener_t` *elp, `eventmask_t` events, `eventflags_t` wflags)

Registers an Event Listener on an Event Source.
- void `chEvtRegisterMaskWithFlags` (`event_source_t` *esp, `event_listener_t` *elp, `eventmask_t` events, `eventflags_t` wflags)

Registers an Event Listener on an Event Source.
- void `chEvtUnregister` (`event_source_t` *esp, `event_listener_t` *elp)

Unregisters an Event Listener from its Event Source.
- `eventmask_t chEvtGetAndClearEventsI` (`eventmask_t` events)

Clears the pending events specified in the events mask.
- `eventmask_t chEvtGetAndClearEvents` (`eventmask_t` events)

Clears the pending events specified in the events mask.
- `eventmask_t chEvtAddEvents` (`eventmask_t` events)

*Adds (OR) a set of events to the current thread, this is **much** faster than using `chEvtBroadcast()` or `chEvtSignal()`.*
- `eventflags_t chEvtGetAndClearFlagsI` (`event_listener_t` *elp)

Returns the unmasked flags associated to an `event_listener_t`.
- `eventflags_t chEvtGetAndClearFlags` (`event_listener_t` *elp)

Returns the flags associated to an `event_listener_t`.
- void `chEvtSignall` (`thread_t` *tp, `eventmask_t` events)

Adds a set of event flags directly to the specified `thread_t`.
- void `chEvtSignal` (`thread_t` *tp, `eventmask_t` events)

Adds a set of event flags directly to the specified `thread_t`.
- void `chEvtBroadcastFlagsI` (`event_source_t` *esp, `eventflags_t` flags)

Signals all the Event Listeners registered on the specified Event Source.
- void `chEvtBroadcastFlags` (`event_source_t` *esp, `eventflags_t` flags)

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

Invokes the event handlers associated to an event flags mask.
- `eventmask_t chEvtWaitOne` (`eventmask_t` events)

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

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

Waits for all the specified events.
- `eventmask_t chEvtWaitOneTimeout` (`eventmask_t` events, `sysinterval_t` timeout)

Waits for exactly one of the specified events.
- `eventmask_t chEvtWaitAnyTimeout` (`eventmask_t` events, `sysinterval_t` timeout)

Waits for any of the specified events.
- `eventmask_t chEvtWaitAllTimeout` (`eventmask_t` events, `sysinterval_t` timeout)

Waits for all the specified events.

9.16.1 Detailed Description

Events code.

9.17 chevents.h File Reference

Events macros and structures.

Data Structures

- struct `event_listener`
Event Listener structure.
- struct `event_source`
Event Source structure.

Macros

- #define `ALL_EVENTS` ((`eventmask_t`)-1)
All events allowed mask.
- #define `EVENT_MASK`(`eid`) ((`eventmask_t`)1 << (`eventmask_t`)(`eid`))
Returns an event mask from an event identifier.
- #define `__EVENTSOURCE_DATA`(`name`) {(`event_listener_t` *)(&`name`)}
Data part of a static event source initializer.
- #define `EVENTSOURCE_DECL`(`name`) `event_source_t` `name` = `__EVENTSOURCE_DATA`(`name`)
Static event source initializer.

Typedefs

- typedef struct `event_source` `event_source_t`
Event Source structure.
- typedef void(* `evhandler_t`) (`eventid_t` `id`)
Event Handler callback function.

Functions

- void `chEvtRegisterMaskWithFlagsI` (`event_source_t` *`esp`, `event_listener_t` *`eip`, `eventmask_t` `events`, `eventflags_t` `wflags`)
Registers an Event Listener on an Event Source.
- void `chEvtRegisterMaskWithFlags` (`event_source_t` *`esp`, `event_listener_t` *`eip`, `eventmask_t` `events`, `eventflags_t` `wflags`)
Registers an Event Listener on an Event Source.
- void `chEvtUnregister` (`event_source_t` *`esp`, `event_listener_t` *`eip`)
Unregisters an Event Listener from its Event Source.
- `eventmask_t chEvtGetAndClearEventsI` (`eventmask_t` `events`)
Clears the pending events specified in the events mask.
- `eventmask_t chEvtGetAndClearEvents` (`eventmask_t` `events`)

- Clears the pending events specified in the events mask.
- `eventmask_t chEvtAddEvents (eventmask_t events)`

Adds (OR) a set of events to the current thread, this is **much** faster than using `chEvtBroadcast ()` or `chEvtSignal ()`.
- `eventflags_t chEvtGetAndClearFlagsI (event_listener_t *elp)`

Returns the unmasked flags associated to an `event_listener_t`.
- `eventflags_t chEvtGetAndClearFlags (event_listener_t *elp)`

Returns the flags associated to an `event_listener_t`.
- `void chEvtSignal (thread_t *tp, eventmask_t events)`

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

Adds a set of event flags directly to the specified `thread_t`.
- `void chEvtBroadcastFlags (event_source_t *esp, eventflags_t flags)`

Signals all the Event Listeners registered on the specified Event Source.
- `void chEvtBroadcastFlagsI (event_source_t *esp, eventflags_t flags)`

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

Invokes the event handlers associated to an event flags mask.
- `eventmask_t chEvtWaitOne (eventmask_t events)`

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

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

Waits for all the specified events.
- `eventmask_t chEvtWaitOneTimeout (eventmask_t events, sysinterval_t timeout)`

Waits for exactly one of the specified events.
- `eventmask_t chEvtWaitAnyTimeout (eventmask_t events, sysinterval_t timeout)`

Waits for any of the specified events.
- `eventmask_t chEvtWaitAllTimeout (eventmask_t events, sysinterval_t timeout)`

Waits for all the specified events.
- `static void chEvtObjectInit (event_source_t *esp)`

Initializes an Event Source.
- `static void chEvtRegisterMask (event_source_t *esp, event_listener_t *elp, eventmask_t events)`

Registers an Event Listener on an Event Source.
- `static void chEvtRegister (event_source_t *esp, event_listener_t *elp, eventid_t event)`

Registers an Event Listener on an Event Source.
- `static bool chEvtIsListeningI (event_source_t *esp)`

Verifies if there is at least one `event_listener_t` registered.
- `static void chEvtBroadcast (event_source_t *esp)`

Signals all the Event Listeners registered on the specified Event Source.
- `static void chEvtBroadcastI (event_source_t *esp)`

Signals all the Event Listeners registered on the specified Event Source.
- `static eventmask_t chEvtAddEventsI (eventmask_t events)`

Adds (OR) a set of events to the current thread, this is **much** faster than using `chEvtBroadcast ()` or `chEvtSignal ()`.
- `static eventmask_t chEvtGetEventsX (void)`

Returns the events mask.

9.17.1 Detailed Description

Events macros and structures.

9.18 chfactory.c File Reference

ChibiOS objects factory and registry code.

```
#include <string.h>
#include "ch.h"
```

Functions

- **void __factory_init (void)**
Initializes the objects factory.
- **registered_object_t * chFactoryRegisterObject (const char *name, void *objp)**
Registers a generic object.
- **registered_object_t * chFactoryFindObject (const char *name)**
Retrieves a registered object.
- **registered_object_t * chFactoryFindObjectByPointer (void *objp)**
Retrieves a registered object by pointer.
- **void chFactoryReleaseObject (registered_object_t *rop)**
Releases a registered object.
- **dyn_buffer_t * chFactoryCreateBuffer (const char *name, size_t size)**
Creates a generic dynamic buffer object.
- **dyn_buffer_t * chFactoryFindBuffer (const char *name)**
Retrieves a dynamic buffer object.
- **void chFactoryReleaseBuffer (dyn_buffer_t *dbp)**
Releases a dynamic buffer object.
- **dyn_semaphore_t * chFactoryCreateSemaphore (const char *name, cnt_t n)**
Creates a dynamic semaphore object.
- **dyn_semaphore_t * chFactoryFindSemaphore (const char *name)**
Retrieves a dynamic semaphore object.
- **void chFactoryReleaseSemaphore (dyn_semaphore_t *dsp)**
Releases a dynamic semaphore object.
- **dyn_mailbox_t * chFactoryCreateMailbox (const char *name, size_t n)**
Creates a dynamic mailbox object.
- **dyn_mailbox_t * chFactoryFindMailbox (const char *name)**
Retrieves a dynamic mailbox object.
- **void chFactoryReleaseMailbox (dyn_mailbox_t *dmp)**
Releases a dynamic mailbox object.
- **dyn_objects_fifo_t * chFactoryCreateObjectsFIFO (const char *name, size_t objsize, size_t objn, unsigned objalign)**
Creates a dynamic "objects FIFO" object.
- **dyn_objects_fifo_t * chFactoryFindObjectsFIFO (const char *name)**
Retrieves a dynamic "objects FIFO" object.
- **void chFactoryReleaseObjectsFIFO (dyn_objects_fifo_t *dofp)**
Releases a dynamic "objects FIFO" object.
- **dyn_pipe_t * chFactoryCreatePipe (const char *name, size_t size)**
Creates a dynamic pipe object.
- **dyn_pipe_t * chFactoryFindPipe (const char *name)**
Retrieves a dynamic pipe object.
- **void chFactoryReleasePipe (dyn_pipe_t *dpp)**
Releases a dynamic pipe object.

Variables

- `objects_factory_t ch_factory`
Factory object static instance.

9.18.1 Detailed Description

ChibiOS objects factory and registry code.

9.19 chfactory.h File Reference

ChibiOS objects factory structures and macros.

Data Structures

- struct `ch_dyn_element`
Type of a dynamic object list element.
- struct `ch_dyn_list`
Type of a dynamic object list.
- struct `ch_registered_static_object`
Type of a registered object.
- struct `ch_dyn_object`
Type of a dynamic buffer object.
- struct `ch_dyn_semaphore`
Type of a dynamic semaphore.
- struct `ch_dyn_mailbox`
Type of a dynamic buffer object.
- struct `ch_dyn_objects_fifo`
Type of a dynamic buffer object.
- struct `ch_dyn_pipe`
Type of a dynamic pipe object.
- struct `ch_objects_factory`
Type of the factory main object.

Macros

- `#define CH_CFG_FACTORY_MAX_NAMES_LENGTH 8`
Maximum length for object names.
- `#define CH_CFG_FACTORY_OBJECTS_REGISTRY TRUE`
Enables the registry of generic objects.
- `#define CH_CFG_FACTORY_GENERIC_BUFFERS TRUE`
Enables factory for generic buffers.
- `#define CH_CFG_FACTORY_SEMAPHORES TRUE`
Enables factory for semaphores.
- `#define CH_CFG_FACTORY_MAILBOXES TRUE`
Enables factory for mailboxes.
- `#define CH_CFG_FACTORY_OBJ_FIFOS TRUE`

- `#define CH_CFG_FACTORY_OBJ_FIFOS TRUE`
Enables factory for objects FIFOs.
- `#define CH_CFG_FACTORY_PIPES TRUE`
Enables factory for Pipes.
- `#define CH_CFG_FACTORY_SEMAPHORES FALSE`
Enables factory for semaphores.
- `#define CH_CFG_FACTORY_MAILBOXES FALSE`
Enables factory for mailboxes.
- `#define CH_CFG_FACTORY_OBJ_FIFOS FALSE`
Enables factory for objects FIFOs.
- `#define CH_CFG_FACTORY_PIPES FALSE`
Enables factory for Pipes.

Typedefs

- `typedef struct ch_dyn_element dyn_element_t`
Type of a dynamic object list element.
- `typedef struct ch_dyn_list dyn_list_t`
Type of a dynamic object list.
- `typedef struct ch_registered_static_object registered_object_t`
Type of a registered object.
- `typedef struct ch_dyn_object dyn_buffer_t`
Type of a dynamic buffer object.
- `typedef struct ch_dyn_semaphore dyn_semaphore_t`
Type of a dynamic semaphore.
- `typedef struct ch_dyn_mailbox dyn_mailbox_t`
Type of a dynamic buffer object.
- `typedef struct ch_dyn_objects_fifo dyn_objects_fifo_t`
Type of a dynamic buffer object.
- `typedef struct ch_dyn_pipe dyn_pipe_t`
Type of a dynamic pipe object.
- `typedef struct ch_objects_factory objects_factory_t`
Type of the factory main object.

Functions

- `void __factory_init (void)`
Initializes the objects factory.
- `registered_object_t * chFactoryRegisterObject (const char *name, void *objp)`
Registers a generic object.
- `registered_object_t * chFactoryFindObject (const char *name)`
Retrieves a registered object.
- `registered_object_t * chFactoryFindObjectByPointer (void *objp)`
Retrieves a registered object by pointer.
- `void chFactoryReleaseObject (registered_object_t *rop)`
Releases a registered object.
- `dyn_buffer_t * chFactoryCreateBuffer (const char *name, size_t size)`
Creates a generic dynamic buffer object.

- `dyn_buffer_t * chFactoryFindBuffer (const char *name)`
Retrieves a dynamic buffer object.
- `void chFactoryReleaseBuffer (dyn_buffer_t *dbp)`
Releases a dynamic buffer object.
- `dyn_semaphore_t * chFactoryCreateSemaphore (const char *name, cnt_t n)`
Creates a dynamic semaphore object.
- `dyn_semaphore_t * chFactoryFindSemaphore (const char *name)`
Retrieves a dynamic semaphore object.
- `void chFactoryReleaseSemaphore (dyn_semaphore_t *dsp)`
Releases a dynamic semaphore object.
- `dyn_mailbox_t * chFactoryCreateMailbox (const char *name, size_t n)`
Creates a dynamic mailbox object.
- `dyn_mailbox_t * chFactoryFindMailbox (const char *name)`
Retrieves a dynamic mailbox object.
- `void chFactoryReleaseMailbox (dyn_mailbox_t *dmp)`
Releases a dynamic mailbox object.
- `dyn_objects_fifo_t * chFactoryCreateObjectsFIFO (const char *name, size_t objsize, size_t objn, unsigned objalign)`
Creates a dynamic "objects FIFO" object.
- `dyn_objects_fifo_t * chFactoryFindObjectsFIFO (const char *name)`
Retrieves a dynamic "objects FIFO" object.
- `void chFactoryReleaseObjectsFIFO (dyn_objects_fifo_t *dofp)`
Releases a dynamic "objects FIFO" object.
- `dyn_pipe_t * chFactoryCreatePipe (const char *name, size_t size)`
Creates a dynamic pipe object.
- `dyn_pipe_t * chFactoryFindPipe (const char *name)`
Retrieves a dynamic pipe object.
- `void chFactoryReleasePipe (dyn_pipe_t *dpp)`
Releases a dynamic pipe object.
- `static dyn_element_t * chFactoryDuplicateReference (dyn_element_t *dep)`
Duplicates an object reference.
- `static void * chFactoryGetObject (registered_object_t *rop)`
Returns the pointer to the inner registered object.
- `static size_t chFactoryGetBufferSize (dyn_buffer_t *dbp)`
Returns the size of a generic dynamic buffer object.
- `static uint8_t * chFactoryGetBuffer (dyn_buffer_t *dbp)`
Returns the pointer to the inner buffer.
- `static semaphore_t * chFactoryGetSemaphore (dyn_semaphore_t *dsp)`
Returns the pointer to the inner semaphore.
- `static mailbox_t * chFactoryGetMailbox (dyn_mailbox_t *dmp)`
Returns the pointer to the inner mailbox.
- `static objects_fifo_t * chFactoryGetObjectsFIFO (dyn_objects_fifo_t *dofp)`
Returns the pointer to the inner objects FIFO.
- `static pipe_t * chFactoryGetPipe (dyn_pipe_t *dpp)`
Returns the pointer to the inner pipe.

9.19.1 Detailed Description

ChibiOS objects factory structures and macros.

9.20 chinstances.c File Reference

OS instances code.

```
#include "ch.h"
```

Functions

- static void `__idle_thread` (void *p)
This function implements the idle thread infinite loop.
- void `chInstanceObjectInit` (`os_instance_t` *oip, const `os_instance_config_t` *oicp)
Initializes a system instance.

9.20.1 Detailed Description

OS instances code.

9.21 chinstances.h File Reference

OS instances macros and structures.

Macros

- #define `__instance_get_currthread`(oip) (oip)->rlist.current
Current thread pointer get macro.
- #define `__instance_set_currthread`(oip, tp) (oip)->rlist.current = (tp)
Current thread pointer set macro.

Functions

- void `chInstanceObjectInit` (`os_instance_t` *oip, const `os_instance_config_t` *oicp)
Initializes a system instance.

9.21.1 Detailed Description

OS instances macros and structures.

9.22 chjobs.h File Reference

Jobs Queues structures and macros.

Data Structures

- struct `ch_jobs_queue`
Type of a jobs queue.
- struct `ch_job_descriptor`
Type of a job descriptor.

Macros

- #define `MSG_JOB_NULL` ((`msg_t`)-2)
Dispatcher return code in case of a `JOB_NUL` has been received.

Typedefs

- typedef struct `ch_jobs_queue` `jobs_queue_t`
Type of a jobs queue.
- typedef void(* `job_function_t`) (`void *arg`)
Type of a job function.
- typedef struct `ch_job_descriptor` `job_descriptor_t`
Type of a job descriptor.

Functions

- static void `chJobObjectInit` (`jobs_queue_t *jqp`, `size_t jobsn`, `job_descriptor_t *jobsbuf`, `msg_t *msgbuf`)
Initializes a jobs queue object.
- static `job_descriptor_t * chJobGet` (`jobs_queue_t *jqp`)
Allocates a free job object.
- static `job_descriptor_t * chJobGetl` (`jobs_queue_t *jqp`)
Allocates a free job object.
- static `job_descriptor_t * chJobGetTimeoutS` (`jobs_queue_t *jqp`, `sysinterval_t timeout`)
Allocates a free job object.
- static `job_descriptor_t * chJobGetTimeout` (`jobs_queue_t *jqp`, `sysinterval_t timeout`)
Allocates a free job object.
- static void `chJobPostl` (`jobs_queue_t *jqp`, `job_descriptor_t *jp`)
Posts a job object.
- static void `chJobPostS` (`jobs_queue_t *jqp`, `job_descriptor_t *jp`)
Posts a job object.
- static void `chJobPost` (`jobs_queue_t *jqp`, `job_descriptor_t *jp`)
Posts a job object.
- static void `chJobPostAheadl` (`jobs_queue_t *jqp`, `job_descriptor_t *jp`)
Posts an high priority job object.
- static void `chJobPostAheadS` (`jobs_queue_t *jqp`, `job_descriptor_t *jp`)
Posts an high priority job object.
- static void `chJobPostAhead` (`jobs_queue_t *jqp`, `job_descriptor_t *jp`)
Posts an high priority job object.
- static `msg_t chJobDispatch` (`jobs_queue_t *jqp`)
Waits for a job then executes it.
- static `msg_t chJobDispatchTimeout` (`jobs_queue_t *jqp`, `sysinterval_t timeout`)
Waits for a job then executes it.

9.22.1 Detailed Description

Jobs Queues structures and macros.

This module implements queues of generic jobs to be delegated asynchronously to a pool of dedicated threads. Operations defined for Jobs Queues

- **Get:** An job object is taken from the pool of the available jobs.
- **Post:** A job is posted to the queue, it will be returned to the pool after execution.

9.23 chlib.h File Reference

ChibiOS/LIB main include file.

```
#include "chbsem.h"
#include "chmboxes.h"
#include "chmemcore.h"
#include "chmemheaps.h"
#include "chmempools.h"
#include "chobjfifos.h"
#include "chpipes.h"
#include "chobjcaches.h"
#include "chdelegates.h"
#include "chjobs.h"
#include "chfactory.h"
```

Macros

- #define **_CHIBIOS_OSLIB_**
ChibiOS/LIB identification macro.
- #define **CH_OSLIB_STABLE** 0
Stable release flag.

ChibiOS/LIB version identification

- #define **CH_OSLIB_VERSION** "1.3.0"
OS Library version string.
- #define **CH_OSLIB_MAJOR** 1
OS Library version major number.
- #define **CH_OSLIB_MINOR** 3
OS Library version minor number.
- #define **CH_OSLIB_PATCH** 0
OS Library version patch number.

Functions

- static void **__oslib_init** (void)
Initialization of all library modules.

9.23.1 Detailed Description

ChibiOS/LIB main include file.

This header includes all the required library headers. This file is meant to be included by `ch.h` not directly by user.

9.24 chlicense.h File Reference

License Module macros and structures.

```
#include "chversion.h"
#include "chcustomer.h"
```

Macros

- `#define CH_LICENSE_TYPE_STRING "GNU General Public License 3 (GPL3)"`
License identification string.
- `#define CH_LICENSE_ID_STRING "N/A"`
Customer identification string.
- `#define CH_LICENSE_ID_CODE "N/A"`
Customer code.
- `#define CH_LICENSE_MODIFIABLE_CODE TRUE`
Code modifiability restrictions.
- `#define CH_LICENSE_FEATURES CH_FEATURES_FULL`
Code functionality restrictions.
- `#define CH_LICENSE_MAX_DEPLOY CH_DEPLOY_UNLIMITED`
Code deploy restrictions.

Allowed Features Levels

- `#define CH_FEATURES_BASIC 0`
- `#define CH_FEATURES_INTERMEDIATE 1`
- `#define CH_FEATURES_FULL 2`

Deployment Options

- `#define CH_DEPLOY_UNLIMITED -1`
- `#define CH_DEPLOY_NONE 0`

Licensing Options

- `#define CH_LICENSE_GPL 0`
- `#define CH_LICENSE_GPL_EXCEPTION 1`
- `#define CH_LICENSE_COMMERCIAL_FREE 2`
- `#define CH_LICENSE_COMMERCIAL_DEV_1000 3`
- `#define CH_LICENSE_COMMERCIAL_DEV_5000 4`
- `#define CH_LICENSE_COMMERCIAL_FULL 5`
- `#define CH_LICENSE_COMMERCIAL_RUNTIME 6`
- `#define CH_LICENSE_PARTNER 7`

9.24.1 Detailed Description

License Module macros and structures.

9.25 chlists.h File Reference

Lists and Queues header.

Data Structures

- struct `ch_list`
Structure representing a generic single link list header and element.
- struct `ch_queue`
Structure representing a generic bidirectional linked list header and element.
- struct `ch_priority_queue`
Structure representing a generic priority-ordered bidirectional linked list header and element.
- struct `ch_delta_list`
Delta list element and header structure.

Macros

- #define `__CH_QUEUE_DATA`(name) `{(ch_queue_t *)&name, (ch_queue_t *)&name}`
Data part of a static queue object initializer.
- #define `CH_QUEUE_DECL`(name) `ch_queue_t name = __CH_QUEUE_DATA(name)`
Static queue object initializer.

Typedefs

- typedef struct `ch_list` `ch_list_t`
Type of a generic single link list header and element.
- typedef struct `ch_queue` `ch_queue_t`
Type of a generic bidirectional linked list header and element.
- typedef struct `ch_priority_queue` `ch_priority_queue_t`
Type of a generic priority-ordered bidirectional linked list header and element.
- typedef struct `ch_delta_list` `ch_delta_list_t`
Type of a generic bidirectional linked delta list header and element.

Functions

- static void `ch_list_init (ch_list_t *lp)`
List initialization.
- static bool `ch_list_isempty (ch_list_t *lp)`
Evaluates to true if the specified list is empty.
- static bool `ch_list_notempty (ch_list_t *lp)`
Evaluates to true if the specified list is not empty.
- static void `ch_list_link (ch_list_t *lp, ch_list_t *p)`
Pushes an element on top of a stack list.
- static `ch_list_t * ch_list_unlink (ch_list_t *lp)`
Pops an element from the top of a stack list and returns it.
- static void `ch_queue_init (ch_queue_t *qp)`
Queue initialization.
- static bool `ch_queue_isempty (const ch_queue_t *qp)`
Evaluates to true if the specified queue is empty.
- static bool `ch_queue_notempty (const ch_queue_t *qp)`
Evaluates to true if the specified queue is not empty.
- static void `ch_queue_insert (ch_queue_t *qp, ch_queue_t *p)`
Inserts an element into a queue.
- static `ch_queue_t * ch_queue_fifo_remove (ch_queue_t *qp)`
Removes the first-out element from a queue and returns it.
- static `ch_queue_t * ch_queue_lifo_remove (ch_queue_t *qp)`
Removes the last-out element from a queue and returns it.
- static `ch_queue_t * ch_queue_dequeue (ch_queue_t *p)`
Removes an element from a queue and returns it.
- static void `ch_pqueue_init (ch_priority_queue_t *pqp)`
Priority queue initialization.
- static `ch_priority_queue_t * ch_pqueue_remove_highest (ch_priority_queue_t *pqp)`
Removes the highest priority element from a priority queue and returns it.
- static `ch_priority_queue_t * ch_pqueue_insert_behind (ch_priority_queue_t *pqp, ch_priority_queue_t *p)`
Inserts an element in the priority queue placing it behind its peers.
- static `ch_priority_queue_t * ch_pqueue_insert_ahead (ch_priority_queue_t *pqp, ch_priority_queue_t *p)`
Inserts an element in the priority queue placing it ahead of its peers.
- static void `ch_dlist_init (ch_delta_list_t *dlhp)`
Delta list initialization.
- static bool `ch_dlist_isempty (ch_delta_list_t *dlhp)`
Evaluates to true if the specified delta list is empty.
- static bool `ch_dlist_notempty (ch_delta_list_t *dlhp)`
Evaluates to true if the specified queue is not empty.
- static bool `ch_dlist_islast (ch_delta_list_t *dlhp, ch_delta_list_t *dlp)`
Last element in the delta list check.
- static bool `ch_dlist_isfirst (ch_delta_list_t *dlhp, ch_delta_list_t *dlp)`
Fist element in the delta list check.
- static void `ch_dlist_insert_after (ch_delta_list_t *dlhp, ch_delta_list_t *dip, sysinterval_t delta)`
Inserts an element after another header element.
- static void `ch_dlist_insert_before (ch_delta_list_t *dlhp, ch_delta_list_t *dip, sysinterval_t delta)`
Inserts an element before another header element.
- static void `ch_dlist_insert (ch_delta_list_t *dlhp, ch_delta_list_t *dip, sysinterval_t delta)`
Inserts an element in a delta list.
- static `ch_delta_list_t * ch_dlist_remove_first (ch_delta_list_t *dlhp)`
Dequeues an element from the delta list.
- static `ch_delta_list_t * ch_dlist_dequeue (ch_delta_list_t *dlp)`
Dequeues an element from the delta list.

9.25.1 Detailed Description

Lists and Queues header.

9.26 chmboxes.c File Reference

Mailboxes code.

```
#include "ch.h"
```

Functions

- void `chMBOBJECTInit (mailbox_t *mbp, msg_t *buf, size_t n)`
Initializes a `mailbox_t` object.
- void `chMBReset (mailbox_t *mbp)`
Resets a `mailbox_t` object.
- void `chMBResetI (mailbox_t *mbp)`
Resets a `mailbox_t` object.
- msg_t `chMBPostTimeout (mailbox_t *mbp, msg_t msg, sysinterval_t timeout)`
Posts a message into a mailbox.
- msg_t `chMBPostTimeoutS (mailbox_t *mbp, msg_t msg, sysinterval_t timeout)`
Posts a message into a mailbox.
- msg_t `chMBPost (mailbox_t *mbp, msg_t msg)`
Posts a message into a mailbox.
- msg_t `chMBPostAheadTimeout (mailbox_t *mbp, msg_t msg, sysinterval_t timeout)`
Posts an high priority message into a mailbox.
- msg_t `chMBPostAheadTimeoutS (mailbox_t *mbp, msg_t msg, sysinterval_t timeout)`
Posts an high priority message into a mailbox.
- msg_t `chMBPostAheadI (mailbox_t *mbp, msg_t msg)`
Posts an high priority message into a mailbox.
- msg_t `chMBFetchTimeout (mailbox_t *mbp, msg_t *msgp, sysinterval_t timeout)`
Retrieves a message from a mailbox.
- msg_t `chMBFetchTimeoutS (mailbox_t *mbp, msg_t *msgp, sysinterval_t timeout)`
Retrieves a message from a mailbox.
- msg_t `chMBFetchI (mailbox_t *mbp, msg_t *msgp)`
Retrieves a message from a mailbox.

9.26.1 Detailed Description

Mailboxes code.

9.27 chmboxes.h File Reference

Mailboxes macros and structures.

Data Structures

- struct `mailbox_t`

Structure representing a mailbox object.

Macros

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

Functions

- `void chMBOBJECTInit (mailbox_t *mbp, msg_t *buf, size_t n)`
Initializes a `mailbox_t` object.
- `void chMBReset (mailbox_t *mbp)`
Resets a `mailbox_t` object.
- `void chMBResetl (mailbox_t *mbp)`
Resets a `mailbox_t` object.
- `msg_t chMBPostTimeout (mailbox_t *mbp, msg_t msg, sysinterval_t timeout)`
Posts a message into a mailbox.
- `msg_t chMBPostTimeoutS (mailbox_t *mbp, msg_t msg, sysinterval_t timeout)`
Posts a message into a mailbox.
- `msg_t chMBPostl (mailbox_t *mbp, msg_t msg)`
Posts a message into a mailbox.
- `msg_t chMBPostAheadTimeout (mailbox_t *mbp, msg_t msg, sysinterval_t timeout)`
Posts an high priority message into a mailbox.
- `msg_t chMBPostAheadTimeoutS (mailbox_t *mbp, msg_t msg, sysinterval_t timeout)`
Posts an high priority message into a mailbox.
- `msg_t chMBPostAheadl (mailbox_t *mbp, msg_t msg)`
Posts an high priority message into a mailbox.
- `msg_t chMBFetchTimeout (mailbox_t *mbp, msg_t *msgp, sysinterval_t timeout)`
Retrieves a message from a mailbox.
- `msg_t chMBFetchTimeoutS (mailbox_t *mbp, msg_t *msgp, sysinterval_t timeout)`
Retrieves a message from a mailbox.
- `msg_t chMBFetchl (mailbox_t *mbp, msg_t *msgp)`
Retrieves a message from a mailbox.
- `static size_t chMBGetSizel (const mailbox_t *mbp)`
Returns the mailbox buffer size as number of messages.
- `static size_t chMBGetUsedCountl (const mailbox_t *mbp)`
Returns the number of used message slots into a mailbox.
- `static size_t chMBGetFreeCountl (const mailbox_t *mbp)`
Returns the number of free message slots into a mailbox.
- `static msg_t chMBPeekl (const mailbox_t *mbp)`
Returns the next message in the queue without removing it.
- `static void chMBResumeX (mailbox_t *mbp)`
Terminates the reset state.

9.27.1 Detailed Description

Mailboxes macros and structures.

9.28 chmemcore.c File Reference

Core memory manager code.

```
#include "ch.h"
```

Functions

- void [__core_init](#) (void)
Low level memory manager initialization.
- void * [chCoreAllocFromBaseI](#) (size_t size, unsigned align, size_t offset)
Allocates a memory block starting from the lowest address upward.
- void * [chCoreAllocFromTopI](#) (size_t size, unsigned align, size_t offset)
Allocates a memory block starting from the top address downward.
- void * [chCoreAllocFromBase](#) (size_t size, unsigned align, size_t offset)
Allocates a memory block starting from the lowest address upward.
- void * [chCoreAllocFromTop](#) (size_t size, unsigned align, size_t offset)
Allocates a memory block starting from the top address downward.
- size_t [chCoreGetStatusX](#) (void)
Core memory status.

Variables

- memcore_t [ch_memcore](#)
Memory core descriptor.

9.28.1 Detailed Description

Core memory manager code.

9.29 chmemcore.h File Reference

Core memory manager macros and structures.

Data Structures

- struct [memcore_t](#)
Type of memory core object.

Macros

- `#define CH_CFG_MEMCORE_SIZE 0`
Managed RAM size.
- `#define chCoreAllocAlignedWithOffset1 chCoreAllocFromTop1`
Allocates a memory block.
- `#define chCoreAllocAlignedWithOffset chCoreAllocFromTop`
Allocates a memory block.

Typedefs

- `typedef void *(* memgetfunc_t) (size_t size, unsigned align)`
Memory get function.
- `typedef void *(* memgetfunc2_t) (size_t size, unsigned align, size_t offset)`
Enhanced memory get function.

Functions

- `void __core_init (void)`
Low level memory manager initialization.
- `void * chCoreAllocFromBase1 (size_t size, unsigned align, size_t offset)`
Allocates a memory block starting from the lowest address upward.
- `void * chCoreAllocFromTop1 (size_t size, unsigned align, size_t offset)`
Allocates a memory block starting from the top address downward.
- `void * chCoreAllocFromBase (size_t size, unsigned align, size_t offset)`
Allocates a memory block starting from the lowest address upward.
- `void * chCoreAllocFromTop (size_t size, unsigned align, size_t offset)`
Allocates a memory block starting from the top address downward.
- `size_t chCoreGetStatusX (void)`
Core memory status.
- `static void * chCoreAllocAligned1 (size_t size, unsigned align)`
Allocates a memory block.
- `static void * chCoreAllocAligned (size_t size, unsigned align)`
Allocates a memory block.
- `static void * chCoreAlloc1 (size_t size)`
Allocates a memory block.
- `static void * chCoreAlloc (size_t size)`
Allocates a memory block.

9.29.1 Detailed Description

Core memory manager macros and structures.

9.30 chmemheaps.c File Reference

Memory heaps code.

```
#include "ch.h"
```

Functions

- void `__heap_init` (void)
Initializes the default heap.
- void `chHeapObjectInit` (`memory_heap_t` *heapp, void *buf, `size_t` size)
Initializes a memory heap from a static memory area.
- void * `chHeapAllocAligned` (`memory_heap_t` *heapp, `size_t` size, unsigned align)
Allocates a block of memory from the heap by using the first-fit algorithm.
- void `chHeapFree` (void *p)
Frees a previously allocated memory block.
- `size_t` `chHeapStatus` (`memory_heap_t` *heapp, `size_t` *totalp, `size_t` *largestp)
Reports the heap status.

Variables

- static `memory_heap_t` `default_heap`
Default heap descriptor.

9.30.1 Detailed Description

Memory heaps code.

9.31 chmemheaps.h File Reference

Memory heaps macros and structures.

Data Structures

- union `heap_header`
Memory heap block header.
- struct `memory_heap`
Structure describing a memory heap.

Macros

- #define `CH_HEAP_ALIGNMENT` 8U
Minimum alignment used for heap.
- #define `CH_HEAP_AREA`(name, size)
Allocation of an aligned static heap buffer.

Typedefs

- typedef struct `memory_heap` `memory_heap_t`
Type of a memory heap.
- typedef union `heap_header` `heap_header_t`
Type of a memory heap header.

Functions

- void `__heap_init` (void)
Initializes the default heap.
- void `chHeapObjectInit` (`memory_heap_t` *heapp, void *buf, `size_t` size)
Initializes a memory heap from a static memory area.
- void * `chHeapAllocAligned` (`memory_heap_t` *heapp, `size_t` size, unsigned align)
Allocates a block of memory from the heap by using the first-fit algorithm.
- void `chHeapFree` (void *p)
Frees a previously allocated memory block.
- `size_t` `chHeapStatus` (`memory_heap_t` *heapp, `size_t` *totalp, `size_t` *largestp)
Reports the heap status.
- static void * `chHeapAlloc` (`memory_heap_t` *heapp, `size_t` size)
Allocates a block of memory from the heap by using the first-fit algorithm.
- static `size_t` `chHeapGetSize` (const void *p)
Returns the size of an allocated block.

9.31.1 Detailed Description

Memory heaps macros and structures.

9.32 chmempools.c File Reference

Memory Pools code.

```
#include "ch.h"
```

Functions

- void `chPoolObjectInitAligned` (`memory_pool_t` *mp, `size_t` size, unsigned align, `memgetfunc_t` provider)
Initializes an empty memory pool.
- void `chPoolLoadArray` (`memory_pool_t` *mp, void *p, `size_t` n)
Loads a memory pool with an array of static objects.
- void * `chPoolAlloc` (`memory_pool_t` *mp)
Allocates an object from a memory pool.
- void * `chPoolAlloc` (`memory_pool_t` *mp)
Allocates an object from a memory pool.
- void `chPoolFree` (`memory_pool_t` *mp, void *objp)
Releases an object into a memory pool.
- void `chPoolFree` (`memory_pool_t` *mp, void *objp)
Releases an object into a memory pool.
- void `chGuardedPoolObjectInitAligned` (`guarded_memory_pool_t` *gmp, `size_t` size, unsigned align)
Initializes an empty guarded memory pool.
- void `chGuardedPoolLoadArray` (`guarded_memory_pool_t` *gmp, void *p, `size_t` n)
Loads a guarded memory pool with an array of static objects.
- void * `chGuardedPoolAllocTimeoutS` (`guarded_memory_pool_t` *gmp, `sysinterval_t` timeout)
Allocates an object from a guarded memory pool.
- void * `chGuardedPoolAllocTimeout` (`guarded_memory_pool_t` *gmp, `sysinterval_t` timeout)
Allocates an object from a guarded memory pool.
- void `chGuardedPoolFree` (`guarded_memory_pool_t` *gmp, void *objp)
Releases an object into a guarded memory pool.

9.32.1 Detailed Description

Memory Pools code.

9.33 chmempools.h File Reference

Memory Pools macros and structures.

Data Structures

- struct `pool_header`
Memory pool free object header.
- struct `memory_pool_t`
Memory pool descriptor.
- struct `guarded_memory_pool_t`
Guarded memory pool descriptor.

Macros

- #define `__MEMORYPOOL_DATA`(name, size, align, provider) {NULL, size, align, provider}
Data part of a static memory pool initializer.
- #define `MEMORYPOOL_DECL`(name, size, align, provider) `memory_pool_t` name = `__MEMORYPOOL_DATA`(name, size, align, provider)
Static memory pool initializer.
- #define `__GUARDEDMEMORYPOOL_DATA`(name, size, align)
Data part of a static guarded memory pool initializer.
- #define `GUARDEDMEMORYPOOL_DECL`(name, size, align) `guarded_memory_pool_t` name = `__GUARDEDMEMORYPOOL_DATA`(name, size, align)
Static guarded memory pool initializer.

Functions

- void `chPoolObjectInitAligned` (`memory_pool_t` *mp, `size_t` size, `unsigned` align, `memgetfunc_t` provider)
Initializes an empty memory pool.
- void `chPoolLoadArray` (`memory_pool_t` *mp, `void` *p, `size_t` n)
Loads a memory pool with an array of static objects.
- `void *` `chPoolAlloc` (`memory_pool_t` *mp)
Allocates an object from a memory pool.
- `void *` `chPoolAlloc` (`memory_pool_t` *mp)
Allocates an object from a memory pool.
- void `chPoolFree` (`memory_pool_t` *mp, `void` *objp)
Releases an object into a memory pool.
- void `chPoolFree` (`memory_pool_t` *mp, `void` *objp)
Releases an object into a memory pool.
- void `chGuardedPoolObjectInitAligned` (`guarded_memory_pool_t` *gmp, `size_t` size, `unsigned` align)
Initializes an empty guarded memory pool.
- void `chGuardedPoolLoadArray` (`guarded_memory_pool_t` *gmp, `void` *p, `size_t` n)

- `void * chGuardedPoolAllocTimeoutS (guarded_memory_pool_t *gmp, sysinterval_t timeout)`
Allocates an object from a guarded memory pool.
- `void * chGuardedPoolAllocTimeout (guarded_memory_pool_t *gmp, sysinterval_t timeout)`
Allocates an object from a guarded memory pool.
- `void chGuardedPoolFree (guarded_memory_pool_t *gmp, void *objp)`
Releases an object into a guarded memory pool.
- `static void chPoolObjectInit (memory_pool_t *mp, size_t size, memgetfunc_t provider)`
Initializes an empty memory pool.
- `static void chPoolAdd (memory_pool_t *mp, void *objp)`
Adds an object to a memory pool.
- `static void chPoolAddl (memory_pool_t *mp, void *objp)`
Adds an object to a memory pool.
- `static void chGuardedPoolObjectInit (guarded_memory_pool_t *gmp, size_t size)`
Initializes an empty guarded memory pool.
- `static cnt_t chGuardedPoolGetCounterl (guarded_memory_pool_t *gmp)`
Gets the count of objects in a guarded memory pool.
- `static void * chGuardedPoolAllocI (guarded_memory_pool_t *gmp)`
Allocates an object from a guarded memory pool.
- `static void chGuardedPoolFreeI (guarded_memory_pool_t *gmp, void *objp)`
Releases an object into a guarded memory pool.
- `static void chGuardedPoolFreeS (guarded_memory_pool_t *gmp, void *objp)`
Releases an object into a guarded memory pool.
- `static void chGuardedPoolAdd (guarded_memory_pool_t *gmp, void *objp)`
Adds an object to a guarded memory pool.
- `static void chGuardedPoolAddl (guarded_memory_pool_t *gmp, void *objp)`
Adds an object to a guarded memory pool.
- `static void chGuardedPoolAddS (guarded_memory_pool_t *gmp, void *objp)`
Adds an object to a guarded memory pool.

9.33.1 Detailed Description

Memory Pools macros and structures.

9.34 chmsg.c File Reference

Messages code.

```
#include "ch.h"
```

Functions

- `msg_t chMsgSend (thread_t *tp, msg_t msg)`
Sends a message to the specified thread.
- `thread_t * chMsgWaitS (void)`
Suspends the thread and waits for an incoming message.
- `thread_t * chMsgWaitTimeoutS (sysinterval_t timeout)`
Suspends the thread and waits for an incoming message or a timeout to occur.
- `thread_t * chMsgPollS (void)`
Poll to check for an incoming message.
- `void chMsgRelease (thread_t *tp, msg_t msg)`
Releases a sender thread specifying a response message.

9.34.1 Detailed Description

Messages code.

9.35 chmsg.h File Reference

Messages macros and structures.

Functions

- `msg_t chMsgSend (thread_t *tp, msg_t msg)`
Sends a message to the specified thread.
- `thread_t * chMsgWaitS (void)`
Suspends the thread and waits for an incoming message.
- `thread_t * chMsgWaitTimeoutS (sysinterval_t timeout)`
Suspends the thread and waits for an incoming message or a timeout to occur.
- `thread_t * chMsgPollS (void)`
Poll to check for an incoming message.
- `void chMsgRelease (thread_t *tp, msg_t msg)`
Releases a sender thread specifying a response message.
- `static thread_t * chMsgWait (void)`
Suspends the thread and waits for an incoming message.
- `static thread_t * chMsgWaitTimeout (sysinterval_t timeout)`
Suspends the thread and waits for an incoming message or a timeout to occur.
- `static thread_t * chMsgPoll (void)`
Poll to check for an incoming message.
- `static bool chMsgIsPendingI (thread_t *tp)`
Evaluates to `true` if the thread has pending messages.
- `static msg_t chMsgGet (thread_t *tp)`
Returns the message carried by the specified thread.
- `static void chMsgReleaseS (thread_t *tp, msg_t msg)`
Releases the thread waiting on top of the messages queue.

9.35.1 Detailed Description

Messages macros and structures.

9.36 chmtx.c File Reference

Mutexes code.

```
#include "ch.h"
```

Functions

- void `chMtxObjectInit` (`mutex_t` *mp)
Initializes a mutex_t structure.
- void `chMtxLock` (`mutex_t` *mp)
Locks the specified mutex.
- void `chMtxLockS` (`mutex_t` *mp)
Locks the specified mutex.
- bool `chMtxTryLock` (`mutex_t` *mp)
Tries to lock a mutex.
- bool `chMtxTryLockS` (`mutex_t` *mp)
Tries to lock a mutex.
- void `chMtxUnlock` (`mutex_t` *mp)
Unlocks the specified mutex.
- void `chMtxUnlockS` (`mutex_t` *mp)
Unlocks the specified mutex.
- void `chMtxUnlockAll` (void)
Unlocks all mutexes owned by the invoking thread.
- void `chMtxUnlockAll` (void)
Unlocks all mutexes owned by the invoking thread.

9.36.1 Detailed Description

Mutexes code.

9.37 chmtx.h File Reference

Mutexes macros and structures.

Data Structures

- struct `ch_mutex`
Mutex structure.

Macros

- #define `__MUTEX_DATA`(name) {`__CH_QUEUE_DATA`(name.queue), NULL, NULL, 0}
Data part of a static mutex initializer.
- #define `MUTEX_DECL`(name) `mutex_t` name = `__MUTEX_DATA`(name)
Static mutex initializer.

Typedefs

- typedef struct `ch_mutex` `mutex_t`
Type of a mutex structure.

Functions

- void `chMtxObjectInit (mutex_t *mp)`
Initializes a mutex_t structure.
- void `chMtxLock (mutex_t *mp)`
Locks the specified mutex.
- void `chMtxLockS (mutex_t *mp)`
Locks the specified mutex.
- bool `chMtxTryLock (mutex_t *mp)`
Tries to lock a mutex.
- bool `chMtxTryLockS (mutex_t *mp)`
Tries to lock a mutex.
- void `chMtxUnlock (mutex_t *mp)`
Unlocks the specified mutex.
- void `chMtxUnlockS (mutex_t *mp)`
Unlocks the specified mutex.
- void `chMtxUnlockAll (void)`
Unlocks all mutexes owned by the invoking thread.
- void `chMtxUnlockAllS (void)`
Unlocks all mutexes owned by the invoking thread.
- static bool `chMtxQueueNotEmptyS (mutex_t *mp)`
Returns true if the mutex queue contains at least a waiting thread.
- static `thread_t * chMtxGetOwnerI (mutex_t *mp)`
Returns the mutex owner thread.
- static `mutex_t * chMtxGetNextMutexX (void)`
Returns the next mutex in the mutexes stack of the current thread.

9.37.1 Detailed Description

Mutexes macros and structures.

9.38 chobjcaches.c File Reference

Objects Caches code.

```
#include "ch.h"
```

Functions

- static `oc_object_t * hash_get_s (objects_cache_t *ocp, uint32_t group, uint32_t key)`
Returns an object pointer from the cache, if present.
- static `oc_object_t * lru_get_last_s (objects_cache_t *ocp)`
Gets the least recently used object buffer from the LRU list.
- void `chCacheObjectInit (objects_cache_t *ocp, ucnt_t hashn, oc_hash_header_t *hashp, ucnt_t objn, size_t objsz, void *objvp, oc_readf_t readf, oc_writef_t writef)`
Initializes a objects_cache_t object.
- `oc_object_t * chCacheGetObject (objects_cache_t *ocp, uint32_t group, uint32_t key)`

- void `chCacheReleaseObject`(`objects_cache_t` *`ocp`, `oc_object_t` *`objp`)
Releases an object into the cache.
- bool `chCacheReadObject`(`objects_cache_t` *`ocp`, `oc_object_t` *`objp`, bool `async`)
Reads object data from the storage.
- bool `chCacheWriteObject`(`objects_cache_t` *`ocp`, `oc_object_t` *`objp`, bool `async`)
Writes the object data back to storage.

9.38.1 Detailed Description

Objects Caches code.

Objects caches.

Operation mode

An object cache allows to retrieve and release objects from a slow media, for example a disk or flash. The most recently used objects are kept in a series of RAM buffers making access faster. Objects are identified by a pair <group, key> which could be mapped, for example, to a disk drive identifier and sector identifier. Read and write operations are performed using externally-supplied functions, the cache is device-agnostic. The cache uses internally an hash table, the size of the table should be dimensioned to minimize the risk of hash collisions, a factor of two is usually acceptable, it depends on the specific application requirements. Operations defined for caches:

- **Get Object:** Retrieves an object from cache, if not present then an empty buffer is returned.
- **Read Object:** Retrieves an object from cache, if not present a buffer is allocated and the object is read from the media.
- **Release Object:** Releases an object to the cache handling the media update, if required.

Precondition

In order to use the pipes APIs the `CH_CFG_USE_OBJ_CACHES` option must be enabled in `chconf.h`.

Note

Compatible with RT and NIL.

9.39 chobjcaches.h File Reference

Objects Caches macros and structures.

Data Structures

- struct `ch_oc_hash_header`
Structure representing an hash table element.
- struct `ch_oc_lru_header`
Structure representing an LRU element header.
- struct `ch_oc_object`
Structure representing a cached object.
- struct `ch_objects_cache`
Structure representing a cache object.

Macros

Cached objects flags

- #define `OC_FLAG_INLRU` 0x00000001U
- #define `OC_FLAG_INHASH` 0x00000002U
- #define `OC_FLAG_SHARED` 0x00000004U
- #define `OC_FLAG_NOTSYNC` 0x00000008U
- #define `OC_FLAG_LAZYWRITE` 0x00000010U
- #define `OC_FLAG_FORGET` 0x00000020U

Typedefs

- typedef uint32_t `oc_flags_t`
Flags of cached objects.
- typedef struct `ch_oc_hash_header` `oc_hash_header_t`
Type of an hash element header.
- typedef struct `ch_oc_lru_header` `oc_lru_header_t`
Type of an LRU element header.
- typedef struct `ch_oc_object` `oc_object_t`
Type of a cached object.
- typedef struct `ch_objects_cache` `objects_cache_t`
Type of a cache object.
- typedef bool(* `oc_readf_t`) (`objects_cache_t` *`ocp`, `oc_object_t` *`objp`, bool `async`)
Object read function.
- typedef bool(* `oc_writef_t`) (`objects_cache_t` *`ocp`, `oc_object_t` *`objp`, bool `async`)
Object write function.

Functions

- void `chCacheObjectInit` (`objects_cache_t` *`ocp`, `ucnt_t` `hashn`, `oc_hash_header_t` *`hashp`, `ucnt_t` `objn`, size_t `objsz`, void *`objvp`, `oc_readf_t` `readf`, `oc_writef_t` `writef`)
Initializes a `objects_cache_t` object.
- `oc_object_t` * `chCacheGetObject` (`objects_cache_t` *`ocp`, uint32_t `group`, uint32_t `key`)
Retrieves an object from the cache.
- void `chCacheReleaseObject` (`objects_cache_t` *`ocp`, `oc_object_t` *`objp`)
Releases an object into the cache.
- bool `chCacheReadObject` (`objects_cache_t` *`ocp`, `oc_object_t` *`objp`, bool `async`)
Reads object data from the storage.
- bool `chCacheWriteObject` (`objects_cache_t` *`ocp`, `oc_object_t` *`objp`, bool `async`)
Writes the object data back to storage.
- static void `chCacheReleaseObject` (`objects_cache_t` *`ocp`, `oc_object_t` *`objp`)
Releases an object into the cache.

9.39.1 Detailed Description

Objects Caches macros and structures.

9.40 chobjects.h File Reference

Operating System Objects macros and structures.

Data Structures

- struct [ch_virtual_timer](#)
Structure representing a Virtual Timer.
- struct [ch_virtual_timers_list](#)
Type of virtual timers list header.
- struct [ch_registry](#)
Type of a registry structure.
- struct [ch_threads_queue](#)
Type of a threads queue.
- struct [ch_thread](#)
Structure representing a thread.
- struct [ch_ready_list](#)
Type of a ready list header.
- struct [ch_os_instance_config](#)
Type of an system instance configuration.
- struct [ch_os_instance](#)
System instance data structure.
- struct [ch_system](#)
Type of system data structure.

Typedefs

- typedef struct [ch_virtual_timer](#) [virtual_timer_t](#)
Type of a Virtual Timer.
- typedef void(* [vtfunc_t](#)) ([virtual_timer_t](#) *vtp, void *p)
Type of a Virtual Timer callback function.
- typedef struct [ch_virtual_timers_list](#) [virtual_timers_list_t](#)
Type of virtual timers list header.
- typedef struct [ch_registry](#) [registry_t](#)
Type of a registry structure.
- typedef [thread_t](#) * [thread_reference_t](#)
Type of a thread reference.
- typedef struct [ch_threads_queue](#) [threads_queue_t](#)
Type of a threads queue.
- typedef struct [ch_ready_list](#) [ready_list_t](#)
Type of a ready list header.
- typedef struct [ch_os_instance_config](#) [os_instance_config_t](#)
Type of an system instance configuration.
- typedef struct [ch_system](#) [ch_system_t](#)
Type of system data structure.

Enumerations

- enum `system_state_t`
Global state of the operating system.

9.40.1 Detailed Description

Operating System Objects macros and structures.

9.41 chobjifos.h File Reference

Objects FIFO structures and macros.

Data Structures

- struct `ch_objects_fifo`
Type of an objects FIFO.

Typedefs

- typedef struct `ch_objects_fifo` `objects_fifo_t`
Type of an objects FIFO.

Functions

- static void `chFifoObjectInitAligned` (`objects_fifo_t` *`ofp`, `size_t` `objszie`, `size_t` `objn`, `unsigned` `objalign`, `void` *`objbuf`, `msg_t` *`msgbuf`)
Initializes a FIFO object.
- static void `chFifoObjectInit` (`objects_fifo_t` *`ofp`, `size_t` `objszie`, `size_t` `objn`, `void` *`objbuf`, `msg_t` *`msgbuf`)
Initializes a FIFO object.
- static void * `chFifoTakeObjectl` (`objects_fifo_t` *`ofp`)
Allocates a free object.
- static void * `chFifoTakeObjectTimeoutS` (`objects_fifo_t` *`ofp`, `sysinterval_t` `timeout`)
Allocates a free object.
- static void * `chFifoTakeObjectTimeout` (`objects_fifo_t` *`ofp`, `sysinterval_t` `timeout`)
Allocates a free object.
- static void `chFifoReturnObjectl` (`objects_fifo_t` *`ofp`, `void` *`objp`)
Releases a fetched object.
- static void `chFifoReturnObjectS` (`objects_fifo_t` *`ofp`, `void` *`objp`)
Releases a fetched object.
- static void `chFifoReturnObject` (`objects_fifo_t` *`ofp`, `void` *`objp`)
Releases a fetched object.
- static void `chFifoSendObjectl` (`objects_fifo_t` *`ofp`, `void` *`objp`)
Posts an object.
- static void `chFifoSendObjectS` (`objects_fifo_t` *`ofp`, `void` *`objp`)
Posts an object.

- static void `chFifoSendObject` (`objects_fifo_t` *`ofp`, `void` *`objp`)
Posts an object.
- static void `chFifoSendObjectAheadI` (`objects_fifo_t` *`ofp`, `void` *`objp`)
Posts an high priority object.
- static void `chFifoSendObjectAheadS` (`objects_fifo_t` *`ofp`, `void` *`objp`)
Posts an high priority object.
- static void `chFifoSendObjectAhead` (`objects_fifo_t` *`ofp`, `void` *`objp`)
Posts an high priority object.
- static `msg_t` `chFifoReceiveObjectI` (`objects_fifo_t` *`ofp`, `void` **`objpp`)
Fetches an object.
- static `msg_t` `chFifoReceiveObjectTimeoutS` (`objects_fifo_t` *`ofp`, `void` **`objpp`, `sysinterval_t` `timeout`)
Fetches an object.
- static `msg_t` `chFifoReceiveObjectTimeout` (`objects_fifo_t` *`ofp`, `void` **`objpp`, `sysinterval_t` `timeout`)
Fetches an object.

9.41.1 Detailed Description

Objects FIFO structures and macros.

This module implements a generic FIFO queue of objects by coupling a Guarded Memory Pool (for objects storage) and a MailBox.

On the sender side free objects are taken from the pool, filled and then sent to the receiver, on the receiver side objects are fetched, used and then returned to the pool. Operations defined for object FIFOs:

- **Take:** An object is taken from the pool of the free objects, can be blocking.
- **Return:** An object is returned to the pool of the free objects, it is guaranteed to be non-blocking.
- **Send:** An object is sent through the mailbox, it is guaranteed to be non-blocking
- **Receive:** An object is received from the mailbox, can be blocking.

9.42 chpipes.c File Reference

Pipes code.

```
#include <string.h>
#include "ch.h"
```

Functions

- static `size_t` `pipe_write` (`pipe_t` *`pp`, `const uint8_t` *`bp`, `size_t` `n`)
Non-blocking pipe write.
- static `size_t` `pipe_read` (`pipe_t` *`pp`, `uint8_t` *`bp`, `size_t` `n`)
Non-blocking pipe read.
- void `chPipeObjectInit` (`pipe_t` *`pp`, `uint8_t` *`buf`, `size_t` `n`)
Initializes a `mailbox_t` object.
- void `chPipeReset` (`pipe_t` *`pp`)
Resets a `pipe_t` object.
- `size_t` `chPipeWriteTimeout` (`pipe_t` *`pp`, `const uint8_t` *`bp`, `size_t` `n`, `sysinterval_t` `timeout`)
Pipe write with timeout.
- `size_t` `chPipeReadTimeout` (`pipe_t` *`pp`, `uint8_t` *`bp`, `size_t` `n`, `sysinterval_t` `timeout`)
Pipe read with timeout.

9.42.1 Detailed Description

Pipes code.

Byte pipes.

Operation mode

A pipe is an asynchronous communication mechanism.

Operations defined for mailboxes:

- **Write:** Writes a buffer of data in the pipe in FIFO order.
- **Read:** A buffer of data is read from the read and removed.
- **Reset:** The pipe is emptied and all the stored data is lost.

Precondition

In order to use the pipes APIs the `CH_CFG_USE_PIPES` option must be enabled in [chconf.h](#).

Note

Compatible with RT and NIL.

9.43 chpipes.h File Reference

Pipes macros and structures.

Data Structures

- struct `pipe_t`

Structure representing a pipe object.

Macros

- `#define __PIPE_DATA(name, buffer, size)`
Data part of a static pipe initializer.
- `#define PIPE_DECL(name, buffer, size) pipe_t name = __PIPE_DATA(name, buffer, size)`
Static pipe initializer.

Functions

- void `chPipeObjectInit` (`pipe_t` *pp, `uint8_t` *buf, `size_t` n)

Initializes a `mailbox_t` object.
- void `chPipeReset` (`pipe_t` *pp)

Resets a `pipe_t` object.
- `size_t chPipeWriteTimeout` (`pipe_t` *pp, const `uint8_t` *bp, `size_t` n, `sysinterval_t` timeout)

Pipe write with timeout.
- `size_t chPipeReadTimeout` (`pipe_t` *pp, `uint8_t` *bp, `size_t` n, `sysinterval_t` timeout)

Pipe read with timeout.
- static `size_t chPipeGetSize` (const `pipe_t` *pp)

Returns the pipe buffer size as number of bytes.
- static `size_t chPipeGetUsedCount` (const `pipe_t` *pp)

Returns the number of used byte slots into a pipe.
- static `size_t chPipeGetFreeCount` (const `pipe_t` *pp)

Returns the number of free byte slots into a pipe.
- static void `chPipeResume` (`pipe_t` *pp)

Terminates the reset state.

9.43.1 Detailed Description

Pipes macros and structures.

9.44 chport.h File Reference

Port wrapper header.

```
#include "chcore.h"
```

9.44.1 Detailed Description

Port wrapper header.

9.45 chregistry.c File Reference

Threads registry code.

```
#include <string.h>
#include "ch.h"
```

Functions

- `thread_t * chRegFirstThread (void)`
Returns the first thread in the system.
- `thread_t * chRegNextThread (thread_t *tp)`
Returns the thread next to the specified one.
- `thread_t * chRegFindThreadByName (const char *name)`
Retrieves a thread pointer by name.
- `thread_t * chRegFindThreadByPointer (thread_t *tp)`
Confirms that a pointer is a valid thread pointer.
- `thread_t * chRegFindThreadByWorkingArea (stkalign_t *wa)`
Confirms that a working area is being used by some active thread.

9.45.1 Detailed Description

Threads registry code.

9.46 chregistry.h File Reference

Threads registry macros and structures.

Data Structures

- struct `chdebug_t`
ChibiOS/RT memory signature record.

Macros

- `#define REG_HEADER(oip) (&ch_system.reclist.queue)`
Access to the registry list header.
- `#define REG_REMOVE(tp) (void) ch_queue_dequeue(&(tp)->rqueue)`
Removes a thread from the registry list.
- `#define REG_INSERT(oip, tp) ch_queue_insert(REG_HEADER(oip), &(tp)->rqueue)`
Adds a thread to the registry list.

Functions

- `thread_t * chRegFirstThread (void)`
Returns the first thread in the system.
- `thread_t * chRegNextThread (thread_t *tp)`
Returns the thread next to the specified one.
- `thread_t * chRegFindThreadByName (const char *name)`
Retrieves a thread pointer by name.
- `thread_t * chRegFindThreadByPointer (thread_t *tp)`
Confirms that a pointer is a valid thread pointer.
- `thread_t * chRegFindThreadByWorkingArea (stkalign_t *wa)`
Confirms that a working area is being used by some active thread.
- `static void __reg_object_init (registry_t *rp)`
Initializes a registry.
- `static void chRegSetThreadName (const char *name)`
Sets the current thread name.
- `static const char * chRegGetThreadNameX (thread_t *tp)`
Returns the name of the specified thread.
- `static void chRegSetThreadNameX (thread_t *tp, const char *name)`
Changes the name of the specified thread.

9.46.1 Detailed Description

Threads registry macros and structures.

9.47 chrestrictions.h File Reference

Licensing restrictions header.

9.47.1 Detailed Description

Licensing restrictions header.

9.48 chrfcu.c File Reference

Runtime Faults Collection Unit code.

```
#include "ch.h"
```

Functions

- `void chRFCUCollectFaultsI (rfcu_mask_t mask)`
Adds fault flags to the current mask.
- `rfcu_mask_t chRFCUGetAndClearFaultsI (rfcu_mask_t mask)`
Returns the current faults mask clearing it.

9.48.1 Detailed Description

Runtime Faults Collection Unit code.

9.49 chrfcu.h File Reference

Runtime Faults Collection Unit macros and structures.

Data Structures

- struct [ch_rfcu](#)
Type of an RFCU structure.

Macros

- #define [CH_RFCU_ALLFAULTS](#) ((rfcu_mask_t)-1)
Mask of all faults.

Predefined Faults

- #define [CH_RFCU_VT_INSUFFICIENT_DELTA](#) 1U
- #define [CH_RFCU_VT_SKIPPED_DEADLINE](#) 2U

Typedefs

- typedef uint32_t [rfcu_mask_t](#)
Type of a faults mask.
- typedef struct [ch_rfcu rfcu_t](#)
Type of an RFCU structure.

Functions

- void [chRFCUCollectFaults](#) (rfcu_mask_t mask)
Adds fault flags to the current mask.
- [rfcu_mask_t chRFCUGetAndClearFaults](#) (rfcu_mask_t mask)
Returns the current faults mask clearing it.
- static void [__rfcu_object_init](#) (rfcu_t *rfcup)
Runtime Faults Collection Unit initialization.

9.49.1 Detailed Description

Runtime Faults Collection Unit macros and structures.

9.50 chschd.c File Reference

Scheduler code.

```
#include "ch.h"
```

Functions

- static `thread_t * __sch_ready_behind (thread_t *tp)`
Inserts a thread in the Ready List placing it behind its peers.
- static `thread_t * __sch_ready_ahead (thread_t *tp)`
Inserts a thread in the Ready List placing it ahead its peers.
- static void `__sch_reschedule_behind (void)`
Switches to the first thread on the runnable queue.
- static void `__sch_reschedule_ahead (void)`
Switches to the first thread on the runnable queue.
- void `ch_sch_prio_insert (ch_queue_t *qp, ch_queue_t *tp)`
Inserts a thread into a priority ordered queue.
- `thread_t * chSchReadyI (thread_t *tp)`
Inserts a thread in the Ready List placing it behind its peers.
- void `chSchGoSleepS (tstate_t newstate)`
Puts the current thread to sleep into the specified state.
- `msg_t chSchGoSleepTimeoutS (tstate_t newstate, sysinterval_t timeout)`
Puts the current thread to sleep into the specified state with timeout specification.
- void `chSchWakeUpS (thread_t *ntp, msg_t msg)`
Wakes up a thread.
- void `chSchRescheduleS (void)`
Performs a reschedule if a higher priority thread is runnable.
- bool `chSchIsPreemptionRequired (void)`
Evaluates if preemption is required.
- void `chSchDoPreemption (void)`
Switches to the first thread on the runnable queue.
- void `chSchPreemption (void)`
All-in-one preemption code.
- void `chSchDoYieldS (void)`
Yields the time slot.
- `thread_t * chSchSelectFirstI (void)`
Makes runnable the fist thread in the ready list, does not reschedule internally.

9.50.1 Detailed Description

Scheduler code.

9.51 chschd.h File Reference

Scheduler macros and structures.

Macros

- `#define firstprio(rlp) ((rlp)->next->prio)`
Returns the priority of the first thread on the given ready list.
- `#define __sch_get_currthread() __instance_get_currthread(currcore)`
Current thread pointer get macro.

Wakeup status codes

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

Priority constants

- `#define NOPRIO (tprio_t)0`
Ready list header priority.
- `#define IDLEPRIO (tprio_t)1`
Idle priority.
- `#define LOWPRIO (tprio_t)2`
Lowest priority.
- `#define NORMALPRIO (tprio_t)128`
Normal priority.
- `#define HIGHPRIO (tprio_t)255`
Highest priority.

Thread states

- `#define CH_STATE_READY (tstate_t)0`
Waiting on the ready list.
- `#define CH_STATE_CURRENT (tstate_t)1`
Currently running.
- `#define CH_STATE_WTSTART (tstate_t)2`
Just created.
- `#define CH_STATE_SUSPENDED (tstate_t)3`
Suspended state.
- `#define CH_STATE_QUEUED (tstate_t)4`
On a queue.
- `#define CH_STATE_WTSEM (tstate_t)5`
On a semaphore.
- `#define CH_STATE_WTMTX (tstate_t)6`

On a mutex.

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

Thread flags and attributes

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

Functions

- `thread_t * chSchReadyI (thread_t *tp)`
Inserts a thread in the Ready List placing it behind its peers.
- `void chSchGoSleepS (tstate_t newstate)`
Puts the current thread to sleep into the specified state.
- `msg_t chSchGoSleepTimeoutS (tstate_t newstate, sysinterval_t timeout)`
Puts the current thread to sleep into the specified state with timeout specification.
- `void chSchWakeupS (thread_t *ntp, msg_t msg)`

- void **chSchRescheduleS** (void)

Wakes up a thread.
- bool **chSchIsPreemptionRequired** (void)

Performs a reschedule if a higher priority thread is runnable.
- void **chSchDoPreemption** (void)

Evaluates if preemption is required.
- void **chSchSelectFirstl** (void)

Switches to the first thread on the runnable queue.
- void **chSchPreemption** (void)

All-in-one preemption code.
- void **chSchDoYieldS** (void)

Yields the time slot.
- **thread_t * chSchInsertFirstl** (void)

Makes runnable the fist thread in the ready list, does not reschedule internally.
- void **ch_sch_prio_insert** (**ch_queue_t** *qp, **ch_queue_t** *tp)

Inserts a thread into a priority ordered queue.

9.51.1 Detailed Description

Scheduler macros and structures.

9.52 chsem.c File Reference

Semaphores code.

```
#include "ch.h"
```

Functions

- void **chSemObjectInit** (**semaphore_t** *sp, **cnt_t** n)

Initializes a semaphore with the specified counter value.
- void **chSemResetWithMessage** (**semaphore_t** *sp, **cnt_t** n, **msg_t** msg)

Performs a reset operation on the semaphore.
- void **chSemResetWithMessageI** (**semaphore_t** *sp, **cnt_t** n, **msg_t** msg)

Performs a reset operation on the semaphore.
- **msg_t chSemWait** (**semaphore_t** *sp)

Performs a wait operation on a semaphore.
- **msg_t chSemWaitS** (**semaphore_t** *sp)

Performs a wait operation on a semaphore.
- **msg_t chSemWaitTimeout** (**semaphore_t** *sp, **sysinterval_t** timeout)

Performs a wait operation on a semaphore with timeout specification.
- **msg_t chSemWaitTimeoutS** (**semaphore_t** *sp, **sysinterval_t** timeout)

Performs a wait operation on a semaphore with timeout specification.
- void **chSemSignal** (**semaphore_t** *sp)

Performs a signal operation on a semaphore.
- void **chSemSignall** (**semaphore_t** *sp)

Performs a signal operation on a semaphore.
- void **chSemAddCounterI** (**semaphore_t** *sp, **cnt_t** n)

Adds the specified value to the semaphore counter.
- **msg_t chSemSignalWait** (**semaphore_t** *sp, **semaphore_t** *spw)

Performs atomic signal and wait operations on two semaphores.

9.52.1 Detailed Description

Semaphores code.

9.53 chsem.h File Reference

Semaphores macros and structures.

Data Structures

- struct `ch_semaphore`
Semaphore structure.

Macros

- #define `__SEMAPHORE_DATA`(name, n) {`__CH_QUEUE_DATA`(name.queue), n}
Data part of a static semaphore initializer.
- #define `SEMAPHORE_DECL`(name, n) `semaphore_t` name = `__SEMAPHORE_DATA`(name, n)
Static semaphore initializer.

Typedefs

- typedef struct `ch_semaphore` `semaphore_t`
Semaphore structure.

Functions

- void `chSemObjectInit` (`semaphore_t` *sp, `cnt_t` n)
Initializes a semaphore with the specified counter value.
- void `chSemResetWithMessage` (`semaphore_t` *sp, `cnt_t` n, `msg_t` msg)
Performs a reset operation on the semaphore.
- void `chSemResetWithMessage1` (`semaphore_t` *sp, `cnt_t` n, `msg_t` msg)
Performs a reset operation on the semaphore.
- `msg_t` `chSemWait` (`semaphore_t` *sp)
Performs a wait operation on a semaphore.
- `msg_t` `chSemWaitS` (`semaphore_t` *sp)
Performs a wait operation on a semaphore.
- `msg_t` `chSemWaitTimeout` (`semaphore_t` *sp, `sysinterval_t` timeout)
Performs a wait operation on a semaphore with timeout specification.
- `msg_t` `chSemWaitTimeoutS` (`semaphore_t` *sp, `sysinterval_t` timeout)
Performs a wait operation on a semaphore with timeout specification.
- void `chSemSignal` (`semaphore_t` *sp)
Performs a signal operation on a semaphore.
- void `chSemSignall` (`semaphore_t` *sp)
Performs a signal operation on a semaphore.
- void `chSemAddCounter1` (`semaphore_t` *sp, `cnt_t` n)

- Adds the specified value to the semaphore counter.
- `msg_t chSemSignalWait (semaphore_t *sps, semaphore_t *spw)`
Performs atomic signal and wait operations on two semaphores.
- `static void chSemReset (semaphore_t *sp, cnt_t n)`
Performs a reset operation on the semaphore.
- `static void chSemResetl (semaphore_t *sp, cnt_t n)`
Performs a reset operation on the semaphore.
- `static void chSemFastWaitl (semaphore_t *sp)`
Decreases the semaphore counter.
- `static void chSemFastSignall (semaphore_t *sp)`
Increases the semaphore counter.
- `static cnt_t chSemGetCounterl (const semaphore_t *sp)`
Returns the semaphore counter current value.

9.53.1 Detailed Description

Semaphores macros and structures.

9.54 chstats.c File Reference

Statistics module code.

```
#include "ch.h"
```

Functions

- `void __stats_increase_irq (void)`
Increases the IRQ counter.
- `void __stats_ctxswc (thread_t *ntp, thread_t *otp)`
Updates context switch related statistics.
- `void __stats_start_measure_crit_thd (void)`
Starts the measurement of a thread critical zone.
- `void __stats_stop_measure_crit_thd (void)`
Stops the measurement of a thread critical zone.
- `void __stats_start_measure_crit_isr (void)`
Starts the measurement of an ISR critical zone.
- `void __stats_stop_measure_crit_isr (void)`
Stops the measurement of an ISR critical zone.

9.54.1 Detailed Description

Statistics module code.

9.55 chstats.h File Reference

Statistics module macros and structures.

Data Structures

- struct `kernel_stats_t`
Type of a kernel statistics structure.

Functions

- void `__stats_increase_irq` (void)
Increases the IRQ counter.
- void `__stats_ctxswc` (`thread_t` *ntp, `thread_t` *otp)
Updates context switch related statistics.
- void `__stats_start_measure_crit_thd` (void)
Starts the measurement of a thread critical zone.
- void `__stats_stop_measure_crit_thd` (void)
Stops the measurement of a thread critical zone.
- void `__stats_start_measure_crit_isr` (void)
Starts the measurement of an ISR critical zone.
- void `__stats_stop_measure_crit_isr` (void)
Stops the measurement of an ISR critical zone.
- static void `__stats_object_init` (`kernel_stats_t` *ksp)
Statistics initialization.

9.55.1 Detailed Description

Statistics module macros and structures.

9.56 chsys.c File Reference

System related code.

```
#include "ch.h"
```

Functions

- static `CH_SYS_CORE0_MEMORY THD_WORKING_AREA` (`ch_c0_idle_thread_wa`, `PORT_IDLE_THREADS`, `AD_STACK_SIZE`)

Working area for core 0 idle thread.
- static `CH_SYS_CORE1_MEMORY THD_WORKING_AREA` (`ch_c1_idle_thread_wa`, `PORT_IDLE_THREADS`, `AD_STACK_SIZE`)

Working area for core 1 idle thread.
- void `chSysWaitSystemState` (`system_state_t state`)

Waits for the system state to be equal to the specified one.
- void `chSysInit` (void)

System initialization.
- void `chSysHalt` (const char *`reason`)

Halts the system.
- bool `chSysIntegrityCheck` (unsigned `testmask`)

System integrity check.
- void `chSysTimerHandler` (void)

Handles time ticks for round robin preemption and timer increments.
- `syssts_t chSysGetStatusAndLockX` (void)

Returns the execution status and enters a critical zone.
- void `chSysRestoreStatusX` (`syssts_t sts`)

Restores the specified execution status and leaves a critical zone.
- bool `chSysIsCounterWithinX` (`rtcnt_t cnt`, `rtcnt_t start`, `rtcnt_t end`)

Realtime window test.
- void `chSysPolledDelayX` (`rtcnt_t cycles`)

Polled delay.

Variables

- `ch_system_t ch_system`

System root object.
- `CH_SYS_CORE0_MEMORY os_instance_t ch0`

Core 0 OS instance.
- const `os_instance_config_t ch_core0_cfg`

Core 0 OS instance configuration.
- `CH_SYS_CORE1_MEMORY os_instance_t ch1`

Core 1 OS instance.
- const `os_instance_config_t ch_core1_cfg`

Core 1 OS instance configuration.

9.56.1 Detailed Description

System related code.

9.57 chsys.h File Reference

System related macros and structures.

Macros

- `#define CH_SYS_CORE0_MEMORY PORT_CORE0_BSS_SECTION`
Core zero memory affinity macro.
- `#define CH_SYS_CORE1_MEMORY PORT_CORE1_BSS_SECTION`
Core one memory affinity macro.
- `#define currcore ch_system.instances[port_get_core_id()]`
Access to current core's instance structure.
- `#define chSysGetRealtimeCounterX() (rtcnt_t)port_rt_get_counter_value()`
Returns the current value of the system real time counter.
- `#define chSysSwitch(ntp, otp)`
Performs a context switch.

Masks of executable integrity checks.

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

ISRs abstraction macros

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

Fast ISRs abstraction macros

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

Time conversion utilities for the realtime counter

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

Functions

- void **chSysWaitSystemState** (*system_state_t* state)
Waits for the system state to be equal to the specified one.
- void **chSysInit** (void)
System initialization.
- bool **chSysIntegrityCheck** (unsigned testmask)
System integrity check.
- void **chSysTimerHandler** (void)
Handles time ticks for round robin preemption and timer increments.
- *syssts_t* **chSysGetStatusAndLockX** (void)
Returns the execution status and enters a critical zone.
- void **chSysRestoreStatusX** (*syssts_t* sts)
Restores the specified execution status and leaves a critical zone.
- bool **chSysIsCounterWithinX** (*rtcnt_t* cnt, *rtcnt_t* start, *rtcnt_t* end)
Realtime window test.
- void **chSysPolledDelayX** (*rtcnt_t* cycles)
Polled delay.
- static void **chSysDisable** (void)
Raises the system interrupt priority mask to the maximum level.
- static void **chSysSuspend** (void)
Raises the system interrupt priority mask to system level.
- static void **chSysEnable** (void)
Lowers the system interrupt priority mask to user level.
- static void **chSysLock** (void)
Enters the kernel lock state.
- static void **chSysUnlock** (void)
Leaves the kernel lock state.
- static void **chSysLockFromISR** (void)
Enters the kernel lock state from within an interrupt handler.
- static void **chSysUnlockFromISR** (void)
Leaves the kernel lock state from within an interrupt handler.
- static void **chSysUnconditionalLock** (void)
Unconditionally enters the kernel lock state.
- static void **chSysUnconditionalUnlock** (void)
Unconditionally leaves the kernel lock state.
- static void **chSysNotifyInstance** (*os_instance_t* *oip)
Notifies an OS instance to check for reschedule.
- static *thread_t* * **chSysGetIdleThreadX** (void)
Returns a pointer to the idle thread.

9.57.1 Detailed Description

System related macros and structures.

9.58 chthreads.c File Reference

Threads code.

```
#include "ch.h"
```

Functions

- `thread_t * __thd_object_init (os_instance_t *oip, thread_t *tp, const char *name, tprio_t prio)`
Initializes a thread structure.
- `void __thd_memfill (uint8_t *startp, uint8_t *endp, uint8_t v)`
Memory fill utility.
- `thread_t * chThdCreateSuspendedI (const thread_descriptor_t *tdp)`
Creates a new thread into a static memory area.
- `thread_t * chThdCreateSuspended (const thread_descriptor_t *tdp)`
Creates a new thread into a static memory area.
- `thread_t * chThdCreateI (const thread_descriptor_t *tdp)`
Creates a new thread into a static memory area.
- `thread_t * chThdCreate (const thread_descriptor_t *tdp)`
Creates a new thread into a static memory area.
- `thread_t * chThdCreateStatic (void *wsp, size_t size, tprio_t prio, tfunc_t pf, void *arg)`
Creates a new thread into a static memory area.
- `thread_t * chThdStart (thread_t *tp)`
Resumes a thread created with `chThdCreateI()`.
- `thread_t * chThdAddRef (thread_t *tp)`
Adds a reference to a thread object.
- `void chThdRelease (thread_t *tp)`
Releases a reference to a thread object.
- `void chThdExit (msg_t msg)`
Terminates the current thread.
- `void chThdExitS (msg_t msg)`
Terminates the current thread.
- `msg_t chThdWait (thread_t *tp)`
Blocks the execution of the invoking thread until the specified thread terminates then the exit code is returned.
- `tprio_t chThdSetPriority (tprio_t newprio)`
Changes the running thread priority level then reschedules if necessary.
- `void chThdTerminate (thread_t *tp)`
Requests a thread termination.
- `void chThdSleep (sysinterval_t time)`
Suspends the invoking thread for the specified time.
- `void chThdSleepUntil (systime_t time)`
Suspends the invoking thread until the system time arrives to the specified value.
- `systime_t chThdSleepUntilWindowed (systime_t prev, systime_t next)`
Suspends the invoking thread until the system time arrives to the specified value.
- `void chThdYield (void)`
Yields the time slot.
- `msg_t chThdSuspendS (thread_reference_t *trp)`
Sends the current thread sleeping and sets a reference variable.
- `msg_t chThdSuspendTimeoutS (thread_reference_t *trp, sysinterval_t timeout)`
Sends the current thread sleeping and sets a reference variable.
- `void chThdResumel (thread_reference_t *trp, msg_t msg)`
Wakes up a thread waiting on a thread reference object.
- `void chThdResumeS (thread_reference_t *trp, msg_t msg)`
Wakes up a thread waiting on a thread reference object.
- `void chThdResume (thread_reference_t *trp, msg_t msg)`
Wakes up a thread waiting on a thread reference object.
- `msg_t chThdEnqueueTimeoutS (threads_queue_t *tqp, sysinterval_t timeout)`

- **void chThdDequeueNextI (threads_queue_t *tqp, msg_t msg)**
Dequeues and wakes up one thread from the threads queue object, if any.
- **void chThdDequeueAllI (threads_queue_t *tqp, msg_t msg)**
Dequeues and wakes up all threads from the threads queue object.

9.58.1 Detailed Description

Threads code.

9.59 chthreads.h File Reference

Threads module macros and structures.

Data Structures

- struct **thread_descriptor_t**
Type of a thread descriptor.

Macros

Threads queues

- #define **__THREADS_QUEUE_DATA**(name) {**_CH_QUEUE_DATA**(name)}
Data part of a static threads queue object initializer.
- #define **THREADS_QUEUE_DECL**(name) **threads_queue_t** name = **__THREADS_QUEUE_DATA**(name)
Static threads queue object initializer.

Working Areas

- #define **THD_WORKING_AREA_SIZE**(n) **MEM_ALIGN_NEXT**(sizeof(**thread_t**) + **PORT_WA_SIZE**(n), **PORT_STACK_ALIGN**)
Calculates the total Working Area size.
- #define **THD_WORKING_AREA**(s, n) **PORT_WORKING_AREA**(s, n)
Static working area allocation.
- #define **THD_WORKING_AREA_BASE**(s) ((**stkalign_t** *) (s))
Base of a working area casted to the correct type.
- #define **THD_WORKING_AREA_END**(s)
End of a working area casted to the correct type.

Threads abstraction macros

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

Threads initializers

- #define **THD_DESCRIPTOR**(name, wbase, wend, prio, funcp, arg)
Thread descriptor initializer with no affinity.

- `#define THD_DESCRIPTOR_AFFINITY(name, wbase, wend, prio, funcp, arg, oip)`
Thread descriptor initializer with no affinity.

Macro Functions

- `#define chThdSleepSeconds(sec) chThdSleep(TIME_S2I(sec))`
Delays the invoking thread for the specified number of seconds.
- `#define chThdSleepMilliseconds(msec) chThdSleep(TIME_MS2I(msec))`
Delays the invoking thread for the specified number of milliseconds.
- `#define chThdSleepMicroseconds(usec) chThdSleep(TIME_US2I(usec))`
Delays the invoking thread for the specified number of microseconds.

TypeDefs

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

Functions

- `thread_t * __thd_object_init (os_instance_t *oip, thread_t *tp, const char *name, tprio_t prio)`
Initializes a thread structure.
- `void __thd_memfill (uint8_t *startp, uint8_t *endp, uint8_t v)`
Memory fill utility.
- `thread_t * chThdCreateSuspendedI (const thread_descriptor_t *tdp)`
Creates a new thread into a static memory area.
- `thread_t * chThdCreateSuspended (const thread_descriptor_t *tdp)`
Creates a new thread into a static memory area.
- `thread_t * chThdCreateI (const thread_descriptor_t *tdp)`
Creates a new thread into a static memory area.
- `thread_t * chThdCreate (const thread_descriptor_t *tdp)`
Creates a new thread into a static memory area.
- `thread_t * chThdCreateStatic (void *wsp, size_t size, tprio_t prio, tfunc_t pf, void *arg)`
Creates a new thread into a static memory area.
- `thread_t * chThdStart (thread_t *tp)`
Resumes a thread created with `chThdCreateI()`.
- `thread_t * chThdAddRef (thread_t *tp)`
Adds a reference to a thread object.
- `void chThdRelease (thread_t *tp)`
Releases a reference to a thread object.
- `void chThdExit (msg_t msg)`
Terminates the current thread.
- `void chThdExitS (msg_t msg)`
Terminates the current thread.
- `msg_t chThdWait (thread_t *tp)`
Blocks the execution of the invoking thread until the specified thread terminates then the exit code is returned.
- `tprio_t chThdSetPriority (tprio_t newprio)`
Changes the running thread priority level then reschedules if necessary.
- `void chThdTerminate (thread_t *tp)`
Requests a thread termination.
- `msg_t chThdSuspendS (thread_reference_t *trp)`

- Sends the current thread sleeping and sets a reference variable.
- `msg_t chThdSuspendTimeoutS (thread_reference_t *trp, sysinterval_t timeout)`
Sends the current thread sleeping and sets a reference variable.
- `void chThdResumel (thread_reference_t *trp, msg_t msg)`
Wakes up a thread waiting on a thread reference object.
- `void chThdResumeS (thread_reference_t *trp, msg_t msg)`
Wakes up a thread waiting on a thread reference object.
- `void chThdResume (thread_reference_t *trp, msg_t msg)`
Wakes up a thread waiting on a thread reference object.
- `msg_t chThdEnqueueTimeoutS (threads_queue_t *tqp, sysinterval_t timeout)`
Enqueues the caller thread on a threads queue object.
- `void chThdDequeueNextl (threads_queue_t *tqp, msg_t msg)`
Dequeues and wakes up one thread from the threads queue object, if any.
- `void chThdDequeueAlll (threads_queue_t *tqp, msg_t msg)`
Dequeues and wakes up all threads from the threads queue object.
- `void chThdSleep (sysinterval_t time)`
Suspends the invoking thread for the specified time.
- `void chThdSleepUntil (systime_t time)`
Suspends the invoking thread until the system time arrives to the specified value.
- `systime_t chThdSleepUntilWindowed (systime_t prev, systime_t next)`
Suspends the invoking thread until the system time arrives to the specified value.
- `void chThdYield (void)`
Yields the time slot.
- `static thread_t * chThdGetSelfX (void)`
Returns a pointer to the current `thread_t`.
- `static tprio_t chThdGetPriorityX (void)`
Returns the current thread priority.
- `static systime_t chThdGetTicksX (thread_t *tp)`
Returns the number of ticks consumed by the specified thread.
- `static stkalign_t * chThdGetWorkingAreaX (thread_t *tp)`
Returns the working area base of the specified thread.
- `static bool chThdTerminatedX (thread_t *tp)`
Verifies if the specified thread is in the `CH_STATE_FINAL` state.
- `static bool chThdShouldTerminateX (void)`
Verifies if the current thread has a termination request pending.
- `static thread_t * chThdStartl (thread_t *tp)`
Resumes a thread created with `chThdCreateI ()`.
- `static void chThdSleepS (sysinterval_t ticks)`
Suspends the invoking thread for the specified number of ticks.
- `static void chThdQueueObjectInit (threads_queue_t *tqp)`
Initializes a threads queue object.
- `static bool chThdQueueIsEmptyl (threads_queue_t *tqp)`
Evaluates to `true` if the specified queue is empty.
- `static void chThdDoDequeueNextl (threads_queue_t *tqp, msg_t msg)`
Dequeues and wakes up one thread from the threads queue object.

9.59.1 Detailed Description

Threads module macros and structures.

9.60 chtime.h File Reference

Time and intervals macros and structures.

Macros

Special time constants

- `#define TIME_IMMEDIATE ((sysinterval_t)0)`
Zero interval specification for some functions with a timeout specification.
- `#define TIME_INFINITE ((sysinterval_t)-1)`
Infinite interval specification for all functions with a timeout specification.
- `#define TIME_MAX_INTERVAL ((sysinterval_t)-2)`
Maximum interval constant usable as timeout.
- `#define TIME_MAX_SYSTIME ((systime_t)-1)`
Maximum system of system time before it wraps.

Fast time conversion utilities

- `#define TIME_S2I(secs) ((sysinterval_t)((time_conv_t)(secs) * (time_conv_t)CH_CFG_ST_FREQUENCY))`
Seconds to time interval.
- `#define TIME_MS2I(msecs)`
Milliseconds to time interval.
- `#define TIME_US2I(usecs)`
Microseconds to time interval.
- `#define TIME_I2S(interval)`
Time interval to seconds.
- `#define TIME_I2MS(interval)`
Time interval to milliseconds.
- `#define TIME_I2US(interval)`
Time interval to microseconds.

Typedefs

- `typedef uint64_t systime_t`
Type of system time.
- `typedef uint64_t sysinterval_t`
Type of time interval.
- `typedef uint64_t systimestamp_t`
Type of a time stamp.
- `typedef uint32_t time_secs_t`
Type of seconds.
- `typedef uint32_t time_msecs_t`
Type of milliseconds.
- `typedef uint32_t time_usecs_t`
Type of microseconds.
- `typedef uint64_t time_conv_t`
Type of time conversion variable.

Functions

Secure time conversion utilities

- static `sysinterval_t chTimeS2I (time_secs_t secs)`
Seconds to time interval.
- static `sysinterval_t chTimeMS2I (time_msecs_t msec)`
Milliseconds to time interval.
- static `sysinterval_t chTimeUS2I (time_usecs_t usec)`
Microseconds to time interval.
- static `time_secs_t chTimel2S (sysinterval_t interval)`
Time interval to seconds.
- static `time_msecs_t chTimel2MS (sysinterval_t interval)`
Time interval to milliseconds.
- static `time_usecs_t chTimel2US (sysinterval_t interval)`
Time interval to microseconds.
- static `systime_t chTimeAddX (systime_t systime, sysinterval_t interval)`
Adds an interval to a system time returning a system time.
- static `sysinterval_t chTimeDiffX (systime_t start, systime_t end)`
Subtracts two system times returning an interval.
- static `bool chTimelsInRangeX (systime_t time, systime_t start, systime_t end)`
Checks if the specified time is within the specified time range.
- static `systimestamp_t chTimeStampAddX (systimestamp_t stamp, sysinterval_t interval)`
Adds an interval to a time stamp returning a time stamp.
- static `sysinterval_t chTimeStampDiffX (systimestamp_t start, systimestamp_t end)`
Subtracts two time stamps returning an interval.
- static `bool chTimeStampIsInRangeX (systimestamp_t stamp, systimestamp_t start, systimestamp_t end)`
Checks if the specified time stamp is within the specified time stamps range.

9.60.1 Detailed Description

Time and intervals macros and structures.

9.61 chtm.c File Reference

Time Measurement module code.

```
#include "ch.h"
```

Functions

- void `chTMOBJECTInit (time_measurement_t *tmp)`
Initializes a TimeMeasurement object.
- NOINLINE void `chTMStartMeasurementX (time_measurement_t *tmp)`
Starts a measurement.
- NOINLINE void `chTMStopMeasurementX (time_measurement_t *tmp)`
Stops a measurement.
- NOINLINE void `chTMChainMeasurementToX (time_measurement_t *tmp1, time_measurement_t *tmp2)`
Stops a measurement and chains to the next one using the same time stamp.

9.61.1 Detailed Description

Time Measurement module code.

9.62 chtm.h File Reference

Time Measurement module macros and structures.

Data Structures

- struct `tm_calibration_t`
Type of a time measurement calibration data.
- struct `time_measurement_t`
Type of a Time Measurement object.

Macros

- `#define TM_CALIBRATION_LOOP 4U`
Number of iterations in the calibration loop.

Functions

- `void chTMOBJECTINIT (time_measurement_t *tmp)`
Initializes a TimeMeasurement object.
- `NOINLINE void chTMStartMeasurementX (time_measurement_t *tmp)`
Starts a measurement.
- `NOINLINE void chTMStopMeasurementX (time_measurement_t *tmp)`
Stops a measurement.
- `NOINLINE void chTMChainMeasurementToX (time_measurement_t *tmp1, time_measurement_t *tmp2)`
Stops a measurement and chains to the next one using the same time stamp.
- `static void __tm_calibration_object_init (tm_calibration_t *tcp)`
Time measurement initialization.

9.62.1 Detailed Description

Time Measurement module macros and structures.

9.63 chtrace.c File Reference

Tracer code.

```
#include "ch.h"
```

Functions

- static NOINLINE void `trace_next` (`os_instance_t` *`oip`)
Writes a time stamp and increases the trace buffer pointer.
- void `__trace_object_init` (`trace_buffer_t` *`tbp`)
Circular trace buffer initialization.
- void `__trace_ready` (`thread_t` *`tp`, `msg_t` `msg`)
Inserts in the circular debug trace buffer a ready record.
- void `__trace_switch` (`thread_t` *`ntp`, `thread_t` *`otp`)
Inserts in the circular debug trace buffer a context switch record.
- void `__trace_isr_enter` (const char *`isr`)
Inserts in the circular debug trace buffer an ISR-enter record.
- void `__trace_isr_leave` (const char *`isr`)
Inserts in the circular debug trace buffer an ISR-leave record.
- void `__trace_halt` (const char *`reason`)
Inserts in the circular debug trace buffer an halt record.
- void `chTraceWritel` (void *`up1`, void *`up2`)
Adds an user trace record to the trace buffer.
- void `chTraceWrite` (void *`up1`, void *`up2`)
Adds an user trace record to the trace buffer.
- void `chTraceSuspendl` (`uint16_t` `mask`)
Suspends one or more trace events.
- void `chTraceSuspend` (`uint16_t` `mask`)
Suspends one or more trace events.
- void `chTraceResumel` (`uint16_t` `mask`)
Resumes one or more trace events.
- void `chTraceResume` (`uint16_t` `mask`)
Resumes one or more trace events.

9.63.1 Detailed Description

Tracer code.

9.64 chtrace.h File Reference

Tracer macros and structures.

Data Structures

- struct `trace_event_t`
Trace buffer record.
- struct `trace_buffer_t`
Trace buffer header.

Macros

Trace record types

- #define **CH_TRACE_TYPE_UNUSED** 0U
- #define **CH_TRACE_TYPE_READY** 1U
- #define **CH_TRACE_TYPE_SWITCH** 2U
- #define **CH_TRACE_TYPE_ISR_ENTER** 3U
- #define **CH_TRACE_TYPE_ISR_LEAVE** 4U
- #define **CH_TRACE_TYPE_HALT** 5U
- #define **CH_TRACE_TYPE_USER** 6U

Events to trace

- #define **CH_DBG_TRACE_MASK_DISABLED** 255U
- #define **CH_DBG_TRACE_MASK_NONE** 0U
- #define **CH_DBG_TRACE_MASK_READY** 1U
- #define **CH_DBG_TRACE_MASK_SWITCH** 2U
- #define **CH_DBG_TRACE_MASK_ISR** 4U
- #define **CH_DBG_TRACE_MASK_HALT** 8U
- #define **CH_DBG_TRACE_MASK_USER** 16U
- #define **CH_DBG_TRACE_MASK_SLOW**
- #define **CH_DBG_TRACE_MASK_ALL**

Debug related settings

- #define **CH_DBG_TRACE_MASK** CH_DBG_TRACE_MASK_DISABLED
Trace buffer entries.
- #define **CH_DBG_TRACE_BUFFER_SIZE** 128
Trace buffer entries.

Functions

- void **__trace_object_init** (trace_buffer_t *tbp)
Circular trace buffer initialization.
- void **__trace_ready** (thread_t *tp, msg_t msg)
Inserts in the circular debug trace buffer a ready record.
- void **__trace_switch** (thread_t *ntp, thread_t *otp)
Inserts in the circular debug trace buffer a context switch record.
- void **__trace_isr_enter** (const char *isr)
Inserts in the circular debug trace buffer an ISR-enter record.
- void **__trace_isr_leave** (const char *isr)
Inserts in the circular debug trace buffer an ISR-leave record.
- void **__trace_halt** (const char *reason)
Inserts in the circular debug trace buffer an halt record.
- void **chTraceWriteI** (void *up1, void *up2)
Adds an user trace record to the trace buffer.
- void **chTraceWrite** (void *up1, void *up2)
Adds an user trace record to the trace buffer.
- void **chTraceSuspendI** (uint16_t mask)
Suspends one or more trace events.
- void **chTraceSuspend** (uint16_t mask)
Suspends one or more trace events.
- void **chTraceResume** (uint16_t mask)
Resumes one or more trace events.

9.64.1 Detailed Description

Tracer macros and structures.

9.65 chversion.h File Reference

Version Module macros and structures.

Macros

- `#define __CHIBIOS__`
ChibiOS product identification macro.
- `#define CH_VERSION_STABLE 1`
Stable release flag.
- `#define CH_VERSION_DATE (((CH_VERSION_YEAR + 2000) * 100) + CH_VERSION_MONTH)`
Current version date in numeric form (yyyymm).

ChibiOS version identification

- `#define CH_VERSION "21.6.0"`
ChibiOS version string.
- `#define CH_VERSION_YEAR 21`
ChibiOS version release year.
- `#define CH_VERSION_MONTH 6`
ChibiOS version release month.
- `#define CH_VERSION_PATCH 0`
ChibiOS version patch number.
- `#define CH_VERSION_NICKNAME "Atrani"`
ChibiOS version nickname.

9.65.1 Detailed Description

Version Module macros and structures.

9.66 chvt.c File Reference

Time and Virtual Timers module code.

```
#include "ch.h"
```

Functions

- static void `vt_insert_first` (`virtual_timers_list_t` *`vtp`, `virtual_timer_t` *`vtp`, `systime_t` `now`, `sysinterval_t` `delay`)
Inserts a timer as first element in a delta list.
- static void `vt_enqueue` (`virtual_timers_list_t` *`vtp`, `virtual_timer_t` *`vtp`, `systime_t` `now`, `sysinterval_t` `delay`)
Enqueues a virtual timer in a virtual timers list.
- void `chVTDoSetl` (`virtual_timer_t` *`vtp`, `sysinterval_t` `delay`, `vtfunc_t` `vtfunc`, `void` *`par`)
Enables a one-shot virtual timer.
- void `chVTDoSetContinuousl` (`virtual_timer_t` *`vtp`, `sysinterval_t` `delay`, `vtfunc_t` `vtfunc`, `void` *`par`)
Enables a continuous virtual timer.
- void `chVTDoResetl` (`virtual_timer_t` *`vtp`)
Disables a Virtual Timer.
- `sysinterval_t chVTGetRemainingIntervall` (`virtual_timer_t` *`vtp`)
Returns the remaining time interval before next timer trigger.
- void `chVTDoTickl` (`void`)
Virtual timers ticker.
- `systimestamp_t chVTGetTimeStampl` (`void`)
Generates a monotonic time stamp.
- void `chVTResetTimeStampl` (`void`)
Resets and re-synchronizes the time stamps monotonic counter.

9.66.1 Detailed Description

Time and Virtual Timers module code.

9.67 chvt.h File Reference

Time and Virtual Timers module macros and structures.

Functions

- void `chVTDoSetl` (`virtual_timer_t` *`vtp`, `sysinterval_t` `delay`, `vtfunc_t` `vtfunc`, `void` *`par`)
Enables a one-shot virtual timer.
- void `chVTDoSetContinuousl` (`virtual_timer_t` *`vtp`, `sysinterval_t` `delay`, `vtfunc_t` `vtfunc`, `void` *`par`)
Enables a continuous virtual timer.
- void `chVTDoResetl` (`virtual_timer_t` *`vtp`)
Disables a Virtual Timer.
- `sysinterval_t chVTGetRemainingIntervall` (`virtual_timer_t` *`vtp`)
Returns the remaining time interval before next timer trigger.
- void `chVTDoTickl` (`void`)
Virtual timers ticker.
- `systimestamp_t chVTGetTimeStampl` (`void`)
Generates a monotonic time stamp.
- void `chVTResetTimeStampl` (`void`)
Resets and re-synchronizes the time stamps monotonic counter.
- static void `chVTOBJECTInit` (`virtual_timer_t` *`vtp`)
Initializes a `virtual_timer_t` object.

- static `systime_t chVTGetSystemTimeX (void)`
Current system time.
- static `systime_t chVTGetSystemTime (void)`
Current system time.
- static `sysinterval_t chVTTIMEElapsedSinceX (systime_t start)`
Returns the elapsed time since the specified start time.
- static `bool chVTIsSystemTimeWithinX (systime_t start, systime_t end)`
Checks if the current system time is within the specified time window.
- static `bool chVTIsSystemTimeWithin (systime_t start, systime_t end)`
Checks if the current system time is within the specified time window.
- static `bool chVTGetTimersStateI (sysinterval_t *timep)`
Returns the time interval until the next timer event.
- static `bool chVTIsArmedI (const virtual_timer_t *vtp)`
Returns true if the specified timer is armed.
- static `bool chVTIsArmed (const virtual_timer_t *vtp)`
Returns true if the specified timer is armed.
- static `void chVTResetI (virtual_timer_t *vtp)`
Disables a Virtual Timer.
- static `void chVTRest (virtual_timer_t *vtp)`
Disables a Virtual Timer.
- static `void chVTSetI (virtual_timer_t *vtp, sysinterval_t delay, vfunc_t vfunc, void *par)`
Enables a one-shot virtual timer.
- static `void chVTSet (virtual_timer_t *vtp, sysinterval_t delay, vfunc_t vfunc, void *par)`
Enables a one-shot virtual timer.
- static `void chVTSetContinuousI (virtual_timer_t *vtp, sysinterval_t delay, vfunc_t vfunc, void *par)`
Enables a continuous virtual timer.
- static `void chVTSetContinuous (virtual_timer_t *vtp, sysinterval_t delay, vfunc_t vfunc, void *par)`
Enables a continuous virtual timer.
- static `sysinterval_t chVTGetReloadIntervalX (virtual_timer_t *vtp)`
Returns the current reload value.
- static `void chVTSetReloadIntervalX (virtual_timer_t *vtp, sysinterval_t reload)`
Changes a timer reload time interval.
- static `systimestamp_t chVTGetTimeStamp (void)`
Generates a monotonic time stamp.
- static `void chVTResetTimeStamp (void)`
Resets and re-synchronizes the time stamps monotonic counter.
- static `__vt_object_init (virtual_timers_list_t *vtlp)`
Virtual Timers instance initialization.

9.67.1 Detailed Description

Time and Virtual Timers module macros and structures.

Index

- __BSEMAPHORE_DATA
 - Binary Semaphores, 354
- __CHIBIOS_OSLIB__
 - Version Numbers and Identification, 350
- __CHIBIOS_RT__
 - Version Numbers and Identification, 29
- __CHIBIOS__
 - Release Information, 22
- __CH_OFFSETOF
 - OS Types and Structures, 60
- __CH_QUEUE_DATA
 - Lists and Queues, 75
- __CH_STRINGIFY
 - OS Types and Structures, 59
- __CH_USED
 - OS Types and Structures, 60
- __CONDVAR_DATA
 - Condition Variables, 262
- __EVENTSOURCE_DATA
 - Event Flags, 275
- __GUARDEDMEMORYPOOL_DATA
 - Memory Pools, 439
- __MAILBOX_DATA
 - Mailboxes, 366
- __MEMORYPOOL_DATA
 - Memory Pools, 438
- __MUTEX_DATA
 - Mutexes, 247
- __PIPE_DATA
 - Pipes, 383
- __SEMAPHORE_DATA
 - Counting Semaphores, 229
- __THREADS_QUEUE_DATA
 - Threads, 183
- __ch_delegate_fn0
 - Delegate Threads, 392
- __ch_delegate_fn1
 - Delegate Threads, 393
- __ch_delegate_fn2
 - Delegate Threads, 393
- __ch_delegate_fn3
 - Delegate Threads, 393
- __ch_delegate_fn4
 - Delegate Threads, 395
- __core_init
 - Core Memory Manager, 422
- __dbg_check_disable
 - Checks and Assertions, 328
- __dbg_check_enable
 - Checks and Assertions, 328
- __dbg_check_enter_isr
 - Checks and Assertions, 332
- __dbg_check_leave_isr
 - Checks and Assertions, 332
- __dbg_check_lock
 - Checks and Assertions, 330
- __dbg_check_lock_from_isr
 - Checks and Assertions, 331
- __dbg_check_suspend
 - Checks and Assertions, 329
- __dbg_check_unlock
 - Checks and Assertions, 330
- __dbg_check_unlock_from_isr
 - Checks and Assertions, 331
- __dbg_object_init
 - Checks and Assertions, 334
- __factory_init
 - Dynamic Objects Factory, 496
- __heap_init
 - Memory Heaps, 432
- __idle_thread
 - OS Instances, 69
- __instance_get_currthread
 - OS Instances, 68
- __instance_set_currthread
 - OS Instances, 68
- __oslib_init
 - Version Numbers and Identification, 351
- __reg_object_init
 - Registry, 321
- __rfcu_object_init
 - Runtime Faults Collection Unit, 73
- __sch_get_currthread
 - Scheduler, 97
- __sch_ready_ahead
 - Scheduler, 99
- __sch_ready_behind
 - Scheduler, 97
- __sch_reschedule_ahead
 - Scheduler, 100
- __sch_reschedule_behind
 - Scheduler, 100
- __stats_ctxswc
 - Statistics, 345
- __stats_increase_irq
 - Statistics, 345
- __stats_object_init
 - Statistics, 347

__stats_start_measure_crit_isr
 Statistics, 347
 __stats_start_measure_crit_thd
 Statistics, 346
 __stats_stop_measure_crit_isr
 Statistics, 347
 __stats_stop_measure_crit_thd
 Statistics, 346
 __thd_memfill
 Threads, 190
 __thd_object_init
 Threads, 189
 __tm_calibration_object_init
 Time Measurement, 225
 __trace_halt
 Tracing, 339
 __trace_isr_enter
 Tracing, 338
 __trace_isr_leave
 Tracing, 339
 __trace_object_init
 Tracing, 337
 __trace_ready
 Tracing, 337
 __trace_switch
 Tracing, 338
 __vt_object_init
 Virtual Timers, 178

align
 memory_pool_t, 597

ALL_EVENTS
 Event Flags, 275

arg
 thread_descriptor_t, 604

Base Kernel Services, 110

basemem
 memcore_t, 593

best
 time_measurement_t, 605

Binary Semaphores, 354
 __BSEMAPHORE_DATA, 354
 binary_semaphore_t, 355
 BSEMAPHORE DECL, 355
 chBSemGetStatel, 363
 chBSemObjectInit, 355
 chBSemReset, 361
 chBSemResetl, 360
 chBSemSignal, 362
 chBSemSignall, 362
 chBSemWait, 356
 chBSemWaitS, 357
 chBSemWaitTimeout, 359
 chBSemWaitTimeoutS, 358

binary_semaphore_t
 Binary Semaphores, 355

BSEMAPHORE DECL
 Binary Semaphores, 355

buf_list
 ch_objects_factory, 537

buffer
 mailbox_t, 591
 pipe_t, 599
 trace_buffer_t, 609

cache_sem
 ch_objects_cache, 534

ch.h, 615

ch0
 System Management, 134

ch1
 System Management, 135

ch_binary_semaphore, 515

CH_CFG_CONTEXT_SWITCH_HOOK
 Options, 51

CH_CFG_FACTORY_GENERIC_BUFFERS
 Dynamic Objects Factory, 493
 Options, 45

CH_CFG_FACTORY_MAILBOXES
 Dynamic Objects Factory, 494
 Options, 46

CH_CFG_FACTORY_MAX_NAMES_LENGTH
 Dynamic Objects Factory, 493
 Options, 45

CH_CFG_FACTORY_OBJ_FIFOS
 Dynamic Objects Factory, 494
 Options, 46

CH_CFG_FACTORY_OBJECTS_REGISTRY
 Dynamic Objects Factory, 493
 Options, 45

CH_CFG_FACTORY_PIPES
 Dynamic Objects Factory, 494, 495
 Options, 46

CH_CFG_FACTORY_SEMAPHORES
 Dynamic Objects Factory, 493
 Options, 45

CH_CFG_IDLE_ENTER_HOOK
 Options, 51

CH_CFG_IDLE_LEAVE_HOOK
 Options, 52

CH_CFG_IDLE_LOOP_HOOK
 Options, 52

CH_CFG_INTERVALS_SIZE
 Options, 36

CH_CFG_IRQ_EPILOGUE_HOOK
 Options, 51

CH_CFG_IRQ_PROLOGUE_HOOK
 Options, 51

CH_CFG_MEMCORE_SIZE
 Core Memory Manager, 420
 Options, 42

CH_CFG_NO_IDLE_THREAD
 Options, 37

CH_CFG_OPTIMIZE_SPEED
 Options, 37

CH_CFG_OS_INSTANCE_EXTRA_FIELDS
 Options, 49

CH_CFG_OS_INSTANCE_INIT_HOOK
 Options, 49

CH_CFG_RUNTIME_FAULTS_HOOK
 Options, 53

CH_CFG_SMP_MODE
 Options, 35

CH_CFG_ST_FREQUENCY
 Options, 36

CH_CFG_ST_RESOLUTION
 Options, 35

CH_CFG_ST_TIMEDELTA
 Options, 36

CH_CFG_SYSTEM_EXTRA_FIELDS
 Options, 49

CH_CFG_SYSTEM_HALT_HOOK
 Options, 53

CH_CFG_SYSTEM_INIT_HOOK
 Options, 49

CH_CFG_SYSTEM_TICK_HOOK
 Options, 52

CH_CFG_THREAD_EXIT_HOOK
 Options, 50

CH_CFG_THREAD_EXTRA_FIELDS
 Options, 50

CH_CFG_THREAD_INIT_HOOK
 Options, 50

CH_CFG_TIME_QUANTUM
 Options, 37

CH_CFG_TIME_TYPES_SIZE
 Options, 36

CH_CFG_TRACE_HOOK
 Options, 53

CH_CFG_USE_CONDVARNS
 Options, 39

CH_CFG_USE_CONDVARNS_TIMEOUT
 Options, 40

CH_CFG_USE_DELEGATES
 Options, 44

CH_CFG_USE_DYNAMIC
 Options, 41

CH_CFG_USE_EVENTS
 Options, 40

CH_CFG_USE_EVENTS_TIMEOUT
 Options, 40

CH_CFG_USE_FACTORY
 Options, 45

CH_CFG_USE_HEAP
 Options, 43

CH_CFG_USE_JOBS
 Options, 44

CH_CFG_USE_MAILBOXES
 Options, 42

CH_CFG_USE_MEMCORE
 Options, 42

CH_CFG_USE_MEMPOOLS
 Options, 43

CH_CFG_USE_MESSAGES
 Options, 41

CH_CFG_USE_MESSAGES_PRIORITY
 Options, 41

CH_CFG_USE_MUTEXES
 Options, 39

CH_CFG_USE_MUTEXES_RECURSIVE
 Options, 39

CH_CFG_USE_OBJ_CACHES
 Options, 44

CH_CFG_USE_OBJ_FIFOS
 Options, 43

CH_CFG_USE_PIPES
 Options, 44

CH_CFG_USE_REGISTRY
 Options, 38

CH_CFG_USE_SEMAPHORES
 Options, 38

CH_CFG_USE_SEMAPHORES_PRIORITY
 Options, 39

CH_CFG_USE_TIMESTAMP
 Options, 38

CH_CFG_USE_TM
 Options, 37

CH_CFG_USE_WAITEXIT
 Options, 38

ch_core0_cfg
 System Management, 135

ch_core1_cfg
 System Management, 135

CH_CUSTOMER_ID_CODE
 Customer Information, 25

CH_CUSTOMER_ID_STRING
 Customer Information, 24

CH_CUSTOMER_LICENSE_EOS_DATE
 Customer Information, 25

CH_CUSTOMER_LICENSE_VERSION_DATE
 Customer Information, 25

CH_CUSTOMER_LICENSE_VERSION_MONTH
 Customer Information, 25

CH_CUSTOMER_LICENSE_VERSION_YEAR
 Customer Information, 25

CH_DBG_ENABLE_ASSERTS
 Options, 47

CH_DBG_ENABLE_CHECKS
 Options, 47

CH_DBG_ENABLE_STACK_CHECK
 Options, 48

CH_DBG_FILL_THREADS
 Options, 48

CH_DBG_STACK_FILL_VALUE
 Checks and Assertions, 327

CH_DBG_STATISTICS
 Options, 46

CH_DBG_SYSTEM_STATE_CHECK
 Options, 46

CH_DBG_THREADS_PROFILING
 Options, 48

CH_DBG_TRACE_BUFFER_SIZE
 Options, 47

Tracing, 336
CH_DBG_TRACE_MASK
 Options, 47
 Tracing, 336
ch_delta_list, 516
 delta, 517
 next, 517
 prev, 517
ch_delta_list_t
 Lists and Queues, 76
ch_dlist_dequeue
 Lists and Queues, 88
ch_dlist_init
 Lists and Queues, 84
ch_dlist_insert
 Lists and Queues, 87
ch_dlist_insert_after
 Lists and Queues, 86
ch_dlist_insert_before
 Lists and Queues, 86
ch_dlist_isempty
 Lists and Queues, 84
ch_dlist_isfirst
 Lists and Queues, 85
ch_dlist_islast
 Lists and Queues, 85
ch_dlist_notempty
 Lists and Queues, 85
ch_dlist_remove_first
 Lists and Queues, 87
ch_dyn_element, 518
 next, 518
 refs, 518
ch_dyn_list, 519
ch_dyn_mailbox, 519
 element, 520
 mbx, 521
ch_dyn_object, 521
 element, 522
ch_dyn_objects_fifo, 522
 element, 523
 fifo, 523
ch_dyn_pipe, 523
 element, 525
 pipe, 525
ch_dyn_semaphore, 525
 element, 526
 sem, 526
ch_factory
 Dynamic Objects Factory, 513
CH_FAST_IRQ_HANDLER
 System Management, 116
CH_FLAG_MODE_HEAP
 Scheduler, 96
CH_FLAG_MODE_MASK
 Scheduler, 96
CH_FLAG_MODE_MPOOL
 Scheduler, 96
CH_FLAG_MODE_STATIC
 Scheduler, 96
CH_FLAG_TERMINATE
 Scheduler, 97
CH_HEAP_ALIGNMENT
 Memory Heaps, 431
CH_HEAP_AREA
 Memory Heaps, 431
CH_IRQ_EPILOGUE
 System Management, 115
CH_IRQ_HANDLER
 System Management, 116
CH_IRQ_IS_VALID_KERNEL_PRIORITY
 System Management, 114
CH_IRQ_IS_VALID_PRIORITY
 System Management, 114
CH_IRQ_PROLOGUE
 System Management, 115
ch_job_descriptor, 526
 jobarg, 527
 jobfunc, 527
ch_jobs_queue, 527
 free, 528
 mbx, 529
CH_KERNEL_MAJOR
 Version Numbers and Identification, 30
CH_KERNEL_MINOR
 Version Numbers and Identification, 30
CH_KERNEL_PATCH
 Version Numbers and Identification, 30
CH_KERNEL_STABLE
 Version Numbers and Identification, 29
CH_KERNEL_VERSION
 Version Numbers and Identification, 30
CH_LICENSE
 Customer Information, 25
CH_LICENSE_FEATURES
 License Settings, 27
CH_LICENSE_ID_CODE
 License Settings, 27
CH_LICENSE_ID_STRING
 License Settings, 27
CH_LICENSE_MAX_DEPLOY
 License Settings, 27
CH_LICENSE_MODIFIABLE_CODE
 License Settings, 27
CH_LICENSE_TYPE_STRING
 License Settings, 26
ch_list, 529
 next, 529
ch_list_init
 Lists and Queues, 77
ch_list_isempty
 Lists and Queues, 77
ch_list_link
 Lists and Queues, 78
ch_list_notempty
 Lists and Queues, 77

ch_list_t
 Lists and Queues, 76

ch_list_unlink
 Lists and Queues, 78

ch_memcore
 Core Memory Manager, 429

ch_mutex, 530
 cnt, 531
 next, 531
 owner, 531
 queue, 531

ch_objects_cache, 532
 cache_sem, 534
 hashn, 533
 hashp, 533
 lru, 534
 lru_sem, 534
 objn, 533
 objsz, 534
 objvp, 534
 readf, 534
 writef, 535

ch_objects_factory, 535
 buf_list, 537
 fifo_list, 537
 mbx_list, 537
 mtx, 536
 obj_list, 536
 obj_pool, 536
 pipe_list, 537
 sem_list, 537
 sem_pool, 537

ch_objects_fifo, 538
 free, 539
 mbx, 539

ch_oc_hash_header, 539
 hash_next, 540
 hash_prev, 541

ch_oc_lru_header, 541
 hash_next, 542
 hash_prev, 542
 lru_next, 542
 lru_prev, 542

ch_oc_object, 543
 dptr, 545
 hash_next, 544
 hash_prev, 544
 lru_next, 544
 lru_prev, 544
 obj_flags, 545
 obj_group, 544
 obj_key, 545
 obj_sem, 545

ch_os_instance, 546
 config, 548
 core_id, 547
 dbg, 548
 kernel_stats, 548

 mainthread, 548
 reglist, 547
 rfcu, 547
 rlist, 547
 trace_buffer, 548
 vlist, 547

ch_os_instance_config, 549
 idlethread_base, 550
 idlethread_end, 550
 mainthread_base, 549
 mainthread_end, 550
 name, 549

CH_OSLIB_MAJOR
 Version Numbers and Identification, 351

CH_OSLIB_MINOR
 Version Numbers and Identification, 351

CH_OSLIB_PATCH
 Version Numbers and Identification, 351

CH_OSLIB_STABLE
 Version Numbers and Identification, 350

CH_OSLIB_VERSION
 Version Numbers and Identification, 350

ch_pqueue_init
 Lists and Queues, 82

ch_pqueue_insert_ahead
 Lists and Queues, 83

ch_pqueue_insert_behind
 Lists and Queues, 83

ch_pqueue_remove_highest
 Lists and Queues, 82

ch_priority_queue, 550
 next, 551
 prev, 551
 prio, 551

ch_priority_queue_t
 Lists and Queues, 76

ch_queue, 552
 next, 552
 prev, 552

CH_QUEUE_DECL
 Lists and Queues, 76

ch_queue_dequeue
 Lists and Queues, 81

ch_queue_fifo_remove
 Lists and Queues, 80

ch_queue_init
 Lists and Queues, 79

ch_queue_insert
 Lists and Queues, 80

ch_queue_isempty
 Lists and Queues, 79

ch_queue_lifo_remove
 Lists and Queues, 81

ch_queue_notempty
 Lists and Queues, 80

ch_queue_t
 Lists and Queues, 76

ch_ready_list, 553

current, 554
 pqueue, 554
ch_registered_static_object, 554
 element, 555
 objp, 555
ch_registry, 556
 queue, 556
ch_rfcu, 557
 mask, 557
CH_RFCU_ALLFAULTS
 Runtime Faults Collection Unit, 71
ch_sch_prio_insert
 Scheduler, 101
ch_semaphore, 558
 cnt, 559
 queue, 559
CH_STATE_CURRENT
 Scheduler, 93
CH_STATE_FINAL
 Scheduler, 95
CH_STATE_NAMES
 Scheduler, 96
CH_STATE_QUEUED
 Scheduler, 94
CH_STATE_READY
 Scheduler, 93
CH_STATE_SLEEPING
 Scheduler, 94
CH_STATE SNDMSG
 Scheduler, 95
CH_STATE SNDMSGQ
 Scheduler, 95
CH_STATE_SUSPENDED
 Scheduler, 93
CH_STATE_WTANDEVT
 Scheduler, 95
CH_STATE_WTCOND
 Scheduler, 94
CH_STATE_WTEXIT
 Scheduler, 94
CH_STATE_WTMMSG
 Scheduler, 95
CH_STATE_WTMTX
 Scheduler, 94
CH_STATE_WTOREVT
 Scheduler, 95
CH_STATE_WTSEM
 Scheduler, 94
CH_STATE_WTSTART
 Scheduler, 93
CH_SYS_CORE0_MEMORY
 System Management, 113
CH_SYS_CORE1_MEMORY
 System Management, 113
ch_system, 559
 instances, 561
 reglist, 561
 rfcu, 561
 state, 560
 System Management, 134
 tmc, 561
ch_system_debug, 562
 isr_cnt, 562
 lock_cnt, 563
 panic_msg, 562
ch_system_t
 OS Types and Structures, 65
ch_thread, 563
 ctx, 566
 epending, 570
 ewmask, 569
 exitcode, 568
 flags, 567
 hdr, 565
 list, 565
 mpool, 571
 msgqueue, 570
 mtxlist, 570
 name, 566
 owner, 566
 pqueue, 565
 queue, 565
 rdymsg, 567
 realprio, 570
 refs, 567
 rqueue, 566
 sentmsg, 568
 state, 566
 stats, 571
 ticks, 567
 time, 567
 u, 569
 wabase, 566
 waiting, 570
 wtmtp, 569
 wtobjp, 568
 wtsemp, 569
 wttrp, 568
ch_threads_queue, 571
 queue, 572
CH_VERSION
 Release Information, 22
CH_VERSION_DATE
 Release Information, 23
CH_VERSION_MONTH
 Release Information, 23
CH_VERSION_NICKNAME
 Release Information, 23
CH_VERSION_PATCH
 Release Information, 23
CH_VERSION_STABLE
 Release Information, 22
CH_VERSION_YEAR
 Release Information, 23
ch_virtual_timer, 572
 dlist, 573

func, 573
par, 573
reload, 573
ch_virtual_timers_list, 574
dlist, 575
laststamp, 575
lasttime, 575
systime, 575
chalign.h, 616
chbsem.h, 617
chBSemGetStateI
 Binary Semaphores, 363
chBSemObjectInit
 Binary Semaphores, 355
chBSemReset
 Binary Semaphores, 361
chBSemResetI
 Binary Semaphores, 360
chBSemSignal
 Binary Semaphores, 362
chBSemSignall
 Binary Semaphores, 362
chBSemWait
 Binary Semaphores, 356
chBSemWaitS
 Binary Semaphores, 357
chBSemWaitTimeout
 Binary Semaphores, 359
chBSemWaitTimeoutS
 Binary Semaphores, 358
chCacheGetObject
 Objects Caches, 485
chCacheObjectInit
 Objects Caches, 484
chCacheReadObject
 Objects Caches, 486
chCacheReleaseObject
 Objects Caches, 488
chCacheReleaseObjectI
 Objects Caches, 486
chCacheWriteObject
 Objects Caches, 487
chchecks.h, 618
chcond.c, 618
chcond.h, 619
chCondBroadcast
 Condition Variables, 265
chCondBroadcastI
 Condition Variables, 266
chCondObjectInit
 Condition Variables, 262
chCondSignal
 Condition Variables, 263
chCondSignall
 Condition Variables, 264
chCondWait
 Condition Variables, 267
chCondWaitS
 Condition Variables, 268
chCondWaitTimeout
 Condition Variables, 269
chCondWaitTimeoutS
 Condition Variables, 270
chconf.h, 620
chCoreAlloc
 Core Memory Manager, 428
chCoreAllocAligned
 Core Memory Manager, 427
chCoreAllocAlignedI
 Core Memory Manager, 426
chCoreAllocAlignedWithOffset
 Core Memory Manager, 421
chCoreAllocAlignedWithOffsetI
 Core Memory Manager, 420
chCoreAllocFromBase
 Core Memory Manager, 424
chCoreAllocFromBaseI
 Core Memory Manager, 422
chCoreAllocFromTop
 Core Memory Manager, 425
chCoreAllocFromTopI
 Core Memory Manager, 423
chCoreAllocI
 Core Memory Manager, 428
chCoreGetStatusX
 Core Memory Manager, 426
chcustomer.h, 623
chDbgAssert
 Checks and Assertions, 327
chDbgCheck
 Checks and Assertions, 327
chDbgCheckClassI
 Checks and Assertions, 333
chDbgCheckClassS
 Checks and Assertions, 333
chdebug.c, 624
chdebug.h, 625
chdebug_t, 576
 identifier, 577
 off_ctx, 578
 off_flags, 579
 off_name, 579
 off_newer, 579
 off_older, 579
 off_preempt, 580
 off_prio, 578
 off_refs, 580
 off_state, 579
 off_stklimit, 579
 off_time, 580
 ptrsize, 578
 size, 577
 threadsize, 578
 timesize, 578
 version, 578
 zero, 577

chDelegateCallDirect0
 Delegate Threads, 397
 chDelegateCallDirect1
 Delegate Threads, 398
 chDelegateCallDirect2
 Delegate Threads, 399
 chDelegateCallDirect3
 Delegate Threads, 400
 chDelegateCallDirect4
 Delegate Threads, 401
 chDelegateCallVeneer
 Delegate Threads, 395
 chDelegateDispatch
 Delegate Threads, 396
 chDelegateDispatchTimeout
 Delegate Threads, 396
 chdelegates.c, 626
 chdelegates.h, 627
 chdynamic.c, 628
 chdynamic.h, 628
 clearly.h, 629
 Checks, 54
 Checks and Assertions, 325
 __dbg_check_disable, 328
 __dbg_check_enable, 329
 __dbg_check_enter_isr, 332
 __dbg_check_leave_isr, 332
 __dbg_check_lock, 330
 __dbg_check_lock_from_isr, 331
 __dbg_check_suspend, 329
 __dbg_check_unlock, 330
 __dbg_check_unlock_from_isr, 331
 __dbg_object_init, 334
 CH_DBG_STACK_FILL_VALUE, 327
 chDbgAssert, 327
 chDbgCheck, 327
 chDbgCheckClassI, 333
 chDbgCheckClassS, 333
 system_debug_t, 328
 chevents.c, 630
 chevents.h, 631
 chEvtAddEvents
 Event Flags, 281
 chEvtAddEventsI
 Event Flags, 298
 chEvtBroadcast
 Event Flags, 297
 chEvtBroadcastFlags
 Event Flags, 287
 chEvtBroadcastFlagsI
 Event Flags, 286
 chEvtBroadcastI
 Event Flags, 297
 chEvtDispatch
 Event Flags, 288
 chEvtGetAndClearEvents
 Event Flags, 280
 chEvtGetAndClearEventsI

 Event Flags, 279
 chEvtGetAndClearFlags
 Event Flags, 283
 chEvtGetAndClearFlagsI
 Event Flags, 282
 chEvtGetEventsX
 Event Flags, 298
 chEvtIsListeningI
 Event Flags, 296
 chEvtObjectInit
 Event Flags, 294
 chEvtRegister
 Event Flags, 295
 chEvtRegisterMask
 Event Flags, 295
 chEvtRegisterMaskWithFlags
 Event Flags, 277
 chEvtRegisterMaskWithFlagsI
 Event Flags, 276
 chEvtSignal
 Event Flags, 285
 chEvtSignall
 Event Flags, 284
 chEvtUnregister
 Event Flags, 278
 chEvtWaitAll
 Event Flags, 290
 chEvtWaitAllTimeout
 Event Flags, 293
 chEvtWaitAny
 Event Flags, 289
 chEvtWaitAnyTimeout
 Event Flags, 292
 chEvtWaitOne
 Event Flags, 288
 chEvtWaitOneTimeout
 Event Flags, 291
 chfactory.c, 633
 chfactory.h, 634
 chFactoryCreateBuffer
 Dynamic Objects Factory, 499
 chFactoryCreateMailbox
 Dynamic Objects Factory, 503
 chFactoryCreateObjectsFIFO
 Dynamic Objects Factory, 504
 chFactoryCreatePipe
 Dynamic Objects Factory, 506
 chFactoryCreateSemaphore
 Dynamic Objects Factory, 501
 chFactoryDuplicateReference
 Dynamic Objects Factory, 509
 chFactoryFindBuffer
 Dynamic Objects Factory, 500
 chFactoryFindMailbox
 Dynamic Objects Factory, 503
 chFactoryFindObject
 Dynamic Objects Factory, 498
 chFactoryFindObjectByPointer

Dynamic Objects Factory, 498
chFactoryFindObjectSIFO
 Dynamic Objects Factory, 505
chFactoryFindPipe
 Dynamic Objects Factory, 507
chFactoryFindSemaphore
 Dynamic Objects Factory, 502
chFactoryGetBuffer
 Dynamic Objects Factory, 510
chFactoryGetBufferSize
 Dynamic Objects Factory, 510
chFactoryGetMailbox
 Dynamic Objects Factory, 511
chFactoryGetObject
 Dynamic Objects Factory, 509
chFactoryGetObjectSIFO
 Dynamic Objects Factory, 512
chFactoryGetPipe
 Dynamic Objects Factory, 512
chFactoryGetSemaphore
 Dynamic Objects Factory, 511
chFactoryRegisterObject
 Dynamic Objects Factory, 497
chFactoryReleaseBuffer
 Dynamic Objects Factory, 501
chFactoryReleaseMailbox
 Dynamic Objects Factory, 504
chFactoryReleaseObject
 Dynamic Objects Factory, 499
chFactoryReleaseObjectsSIFO
 Dynamic Objects Factory, 506
chFactoryReleasePipe
 Dynamic Objects Factory, 507
chFactoryReleaseSemaphore
 Dynamic Objects Factory, 502
chFifoObjectInit
 Objects FIFOs, 465
chFifoObjectInitAligned
 Objects FIFOs, 464
chFifoReceiveObjectI
 Objects FIFOs, 476
chFifoReceiveObjectTimeout
 Objects FIFOs, 478
chFifoReceiveObjectTimeoutS
 Objects FIFOs, 477
chFifoReturnObject
 Objects FIFOs, 470
chFifoReturnObjectI
 Objects FIFOs, 469
chFifoReturnObjectS
 Objects FIFOs, 469
chFifoSendObject
 Objects FIFOs, 473
chFifoSendObjectAhead
 Objects FIFOs, 476
chFifoSendObjectAheadI
 Objects FIFOs, 474
chFifoSendObjectAheadS
 Objects FIFOs, 475
chFifoSendObjectI
 Objects FIFOs, 471
chFifoSendObjectS
 Objects FIFOs, 472
chFifoTakeObjectI
 Objects FIFOs, 466
chFifoTakeObjectTimeout
 Objects FIFOs, 468
chFifoTakeObjectTimeoutS
 Objects FIFOs, 467
chGuardedPoolAdd
 Memory Pools, 458
chGuardedPoolAddI
 Memory Pools, 459
chGuardedPoolAddS
 Memory Pools, 460
chGuardedPoolAlloc
 Memory Pools, 455
chGuardedPoolAllocTimeout
 Memory Pools, 449
chGuardedPoolAllocTimeoutS
 Memory Pools, 448
chGuardedPoolFree
 Memory Pools, 450
chGuardedPoolFreeI
 Memory Pools, 456
chGuardedPoolFreeS
 Memory Pools, 457
chGuardedPoolGetCounterI
 Memory Pools, 454
chGuardedPoolLoadArray
 Memory Pools, 447
chGuardedPoolObjectInit
 Memory Pools, 454
chGuardedPoolObjectInitAligned
 Memory Pools, 446
chHeapAlloc
 Memory Heaps, 434
chHeapAllocAligned
 Memory Heaps, 433
chHeapFree
 Memory Heaps, 433
chHeapGetSize
 Memory Heaps, 435
chHeapObjectInit
 Memory Heaps, 432
chHeapStatus
 Memory Heaps, 434
chInstanceObjectInit
 OS Instances, 69
chinstances.c, 637
chinstances.h, 637
chJobDispatch
 Jobs Queues, 415
chJobDispatchTimeout
 Jobs Queues, 416
chJobGet

Jobs Queues, 405
 chJobGetI
 Jobs Queues, 406
 chJobGetTimeout
 Jobs Queues, 408
 chJobGetTimeoutS
 Jobs Queues, 407
 chJobObjectInit
 Jobs Queues, 404
 chJobPost
 Jobs Queues, 411
 chJobPostAhead
 Jobs Queues, 414
 chJobPostAheadI
 Jobs Queues, 412
 chJobPostAheadS
 Jobs Queues, 413
 chJobPostI
 Jobs Queues, 409
 chJobPostS
 Jobs Queues, 410
 chjobs.h, 637
 chlib.h, 639
 chlicense.h, 640
 chlists.h, 641
 chMBFetchI
 Mailboxes, 378
 chMBFetchTimeout
 Mailboxes, 376
 chMBFetchTimeoutS
 Mailboxes, 377
 chMBGetFreeCountI
 Mailboxes, 380
 chMBGetSizeI
 Mailboxes, 379
 chMBGetUsedCountI
 Mailboxes, 380
 chMBOBJECTInit
 Mailboxes, 367
 chmboxes.c, 643
 chmboxes.h, 643
 chMBPeekI
 Mailboxes, 381
 chMBPostAheadI
 Mailboxes, 375
 chMBPostAheadTimeout
 Mailboxes, 373
 chMBPostAheadTimeoutS
 Mailboxes, 374
 chMBPostI
 Mailboxes, 372
 chMBPostTimeout
 Mailboxes, 370
 chMBPostTimeoutS
 Mailboxes, 371
 chMBReset
 Mailboxes, 368
 chMBResetI

Mailboxes, 369
 chMBResumeX
 Mailboxes, 382
 chmemcore.c, 645
 chmemcore.h, 645
 chmemheaps.c, 646
 chmemheaps.h, 647
 chmempools.c, 648
 chmempools.h, 649
 chmsg.c, 650
 chmsg.h, 651
 chMsgGet
 Synchronous Messages, 309
 chMsgIsPendingI
 Synchronous Messages, 308
 chMsgPoll
 Synchronous Messages, 307
 chMsgPollS
 Synchronous Messages, 303
 chMsgRelease
 Synchronous Messages, 304
 chMsgReleaseS
 Synchronous Messages, 310
 chMsgSend
 Synchronous Messages, 301
 chMsgWait
 Synchronous Messages, 305
 chMsgWaitS
 Synchronous Messages, 301
 chMsgWaitTimeout
 Synchronous Messages, 306
 chMsgWaitTimeoutS
 Synchronous Messages, 302
 chmtx.c, 651
 chmtx.h, 652
 chMtxGetNextMutexX
 Mutexes, 260
 chMtxGetOwnerI
 Mutexes, 259
 chMtxLock
 Mutexes, 250
 chMtxLockS
 Mutexes, 251
 chMtxObjectInit
 Mutexes, 249
 chMtxQueueNotEmptyS
 Mutexes, 258
 chMtxTryLock
 Mutexes, 252
 chMtxTryLockS
 Mutexes, 253
 chMtxUnlock
 Mutexes, 254
 chMtxUnlockAll
 Mutexes, 257
 chMtxUnlockAllS
 Mutexes, 256
 chMtxUnlockS

Mutexes, 255
chobjcaches.c, 653
chobjcaches.h, 654
chobjects.h, 656
chobjifos.h, 657
chPipeGetFreeCount
 Pipes, 389
chPipeGetSize
 Pipes, 388
chPipeGetUsedCount
 Pipes, 388
chPipeObjectInit
 Pipes, 386
chPipeReadTimeout
 Pipes, 387
chPipeReset
 Pipes, 386
chPipeResume
 Pipes, 389
chpipes.c, 658
chpipes.h, 659
chPipeWriteTimeout
 Pipes, 386
chPoolAdd
 Memory Pools, 452
chPoolAddl
 Memory Pools, 453
chPoolAlloc
 Memory Pools, 443
chPoolAllocl
 Memory Pools, 442
chPoolFree
 Memory Pools, 445
chPoolFreel
 Memory Pools, 444
chPoolLoadArray
 Memory Pools, 441
chPoolObjectInit
 Memory Pools, 451
chPoolObjectInitAligned
 Memory Pools, 440
chport.h, 660
chRegFindThreadByName
 Registry, 318
chRegFindThreadByPointer
 Registry, 319
chRegFindThreadByWorkingArea
 Registry, 320
chRegFirstThread
 Registry, 316
chRegGetThreadNameX
 Registry, 322
chregistry.c, 660
chregistry.h, 661
chRegNextThread
 Registry, 317
chRegSetThreadName
 Registry, 321
chRegSetThreadNameX
 Registry, 322
chrestrictions.h, 662
chrfcu.c, 662
chrfcu.h, 663
chRFCUCollectFaultsI
 Runtime Faults Collection Unit, 72
chRFCUGetAndClearFaultsI
 Runtime Faults Collection Unit, 72
chsched.c, 664
chsched.h, 664
chSchDoPreemption
 Scheduler, 107
chSchDoYieldS
 Scheduler, 108
chSchGoSleepS
 Scheduler, 103
chSchGoSleepTimeoutS
 Scheduler, 104
chSchIsPreemptionRequired
 Scheduler, 106
chSchPreemption
 Scheduler, 108
chSchReadyI
 Scheduler, 102
chSchRescheduleS
 Scheduler, 106
chSchSelectFirstI
 Scheduler, 109
chSchWakeups
 Scheduler, 105
chsem.c, 667
chsem.h, 668
chSemAddCounterI
 Counting Semaphores, 239
chSemFastSignall
 Counting Semaphores, 243
chSemFastWaitI
 Counting Semaphores, 243
chSemGetCounterI
 Counting Semaphores, 244
chSemObjectInit
 Counting Semaphores, 230
chSemReset
 Counting Semaphores, 241
chSemResetI
 Counting Semaphores, 242
chSemResetWithMessage
 Counting Semaphores, 231
chSemResetWithMessageI
 Counting Semaphores, 232
chSemSignal
 Counting Semaphores, 237
chSemSignall
 Counting Semaphores, 238
chSemSignalWait
 Counting Semaphores, 240
chSemWait

Counting Semaphores, 233
chSemWaitS
 Counting Semaphores, 234
chSemWaitTimeout
 Counting Semaphores, 235
chSemWaitTimeoutS
 Counting Semaphores, 236
chstats.c, 669
chstats.h, 670
chsys.c, 670
chsys.h, 671
chSysDisable
 System Management, 128
chSysEnable
 System Management, 129
chSysGetIdleThreadX
 System Management, 134
chSysGetRealtimeCounterX
 System Management, 120
chSysGetStatusAndLockX
 System Management, 125
chSysHalt
 OS Types and Structures, 66
 System Management, 122
chSysInit
 System Management, 122
chSysIntegrityCheckI
 System Management, 123
chSysIsCounterWithinX
 System Management, 127
chSysLock
 System Management, 130
chSysLockFromISR
 System Management, 131
chSysNotifyInstance
 System Management, 133
chSysPolledDelayX
 System Management, 128
chSysRestoreStatusX
 System Management, 126
chSysSuspend
 System Management, 129
chSysSwitch
 System Management, 120
chSysTimerHandlerI
 System Management, 124
chSysUnconditionalLock
 System Management, 132
chSysUnconditionalUnlock
 System Management, 133
chSysUnlock
 System Management, 130
chSysUnlockFromISR
 System Management, 132
chSysWaitSystemState
 System Management, 121
chThdAddRef
 Threads, 197
chThdCreate
 Threads, 193
chThdCreateFromHeap
 Dynamic Threads, 311
chThdCreateFromMemoryPool
 Dynamic Threads, 312
chThdCreateI
 Threads, 192
chThdCreateStatic
 Threads, 195
chThdCreateSuspended
 Threads, 191
chThdCreateSuspendedI
 Threads, 190
chThdDequeueAllI
 Threads, 215
chThdDequeueNextI
 Threads, 214
chThdDoDequeueNextI
 Threads, 221
chThdEnqueueTimeoutS
 Threads, 213
chThdExit
 Threads, 199
chThdExitS
 Threads, 200
chThdGetPriorityX
 Threads, 216
chThdGetSelfX
 Threads, 216
chThdGetTicksX
 Threads, 216
chThdGetWorkingAreaX
 Threads, 217
chThdQueueIsEmptyI
 Threads, 220
chThdQueueObjectInit
 Threads, 220
chThdRelease
 Threads, 198
chThdResume
 Threads, 212
chThdResumel
 Threads, 210
chThdResumeS
 Threads, 211
chThdSetPriority
 Threads, 202
chThdShouldTerminateX
 Threads, 218
chThdSleep
 Threads, 204
chThdSleepMicroseconds
 Threads, 188
chThdSleepMilliseconds
 Threads, 188
chThdSleepS
 Threads, 219

chThdSleepSeconds
 Threads, 187
chThdSleepUntil
 Threads, 205
chThdSleepUntilWindowed
 Threads, 206
chThdStart
 Threads, 196
chThdStartI
 Threads, 218
chThdSuspendS
 Threads, 208
chThdSuspendTimeoutS
 Threads, 209
chThdTerminate
 Threads, 203
chThdTerminatedX
 Threads, 217
chThdWait
 Threads, 201
chThdYield
 Threads, 207
chthreads.c, 673
chthreads.h, 675
ctime.h, 678
chTimeAddX
 Time and Intervals, 149
chTimeDiffX
 Time and Intervals, 150
chTimeI2MS
 Time and Intervals, 148
chTimeI2S
 Time and Intervals, 148
chTimeI2US
 Time and Intervals, 149
chTimeIsInRangeX
 Time and Intervals, 150
chTimeMS2I
 Time and Intervals, 147
chTimeS2I
 Time and Intervals, 146
chTimeStampAddX
 Time and Intervals, 151
chTimeStampDiffX
 Time and Intervals, 151
chTimeStampIsInRangeX
 Time and Intervals, 152
chTimeUS2I
 Time and Intervals, 147
cthm.c, 679
cthm.h, 680
chTMChainMeasurementToX
 Time Measurement, 225
chTMOBJECTInit
 Time Measurement, 223
chTMStartMeasurementX
 Time Measurement, 224
chTMStopMeasurementX
 Time Measurement, 224
chtrace.c, 680
chtrace.h, 681
chTraceResume
 Tracing, 343
chTraceResumel
 Tracing, 343
chTraceSuspend
 Tracing, 342
chTraceSuspendl
 Tracing, 341
chTraceWrite
 Tracing, 340
chTraceWritel
 Tracing, 339
chversion.h, 683
chvt.c, 683
chvt.h, 684
chVTDoResetI
 Virtual Timers, 157
chVTDoSetContinuousI
 Virtual Timers, 156
chVTDoSetI
 Virtual Timers, 155
chVTDoTickI
 Virtual Timers, 159
chVTGetReloadIntervalX
 Virtual Timers, 175
chVTGetRemainingIntervalI
 Virtual Timers, 158
chVTGetSystemTime
 Virtual Timers, 162
chVTGetSystemTimeX
 Virtual Timers, 162
chVTGetTimersStateI
 Virtual Timers, 166
chVTGetTimeStamp
 Virtual Timers, 176
chVTGetTimeStampI
 Virtual Timers, 160
chVTIsArmed
 Virtual Timers, 168
chVTIsArmedI
 Virtual Timers, 167
chVTIsSystemTimeWithin
 Virtual Timers, 165
chVTIsSystemTimeWithinX
 Virtual Timers, 164
chVTOBJECTInit
 Virtual Timers, 161
chVTRestore
 Virtual Timers, 170
chVTRestoreI
 Virtual Timers, 169
chVTRestoreTimeStamp
 Virtual Timers, 177
chVTRestoreTimeStampI
 Virtual Timers, 161

chVTSet
 Virtual Timers, 172
 chVTSetContinuous
 Virtual Timers, 174
 chVTSetContinuousI
 Virtual Timers, 173
 chVTSetI
 Virtual Timers, 171
 chVTSetReloadIntervalX
 Virtual Timers, 175
 chVTTTimeElapsedSinceX
 Virtual Timers, 163
 cmtx
 pipe_t, 600
 cnt
 ch_mutex, 531
 ch_semaphore, 559
 mailbox_t, 592
 pipe_t, 600
 cnt_t
 OS Types and Structures, 63
 Complex Services, 462
 Condition Variables, 261
 __CONDVAR_DATA, 262
 chCondBroadcast, 265
 chCondBroadcastI, 266
 chCondObjectInit, 262
 chCondSignal, 263
 chCondSignall, 264
 chCondWait, 267
 chCondWaitS, 268
 chCondWaitTimeout, 269
 chCondWaitTimeoutS, 270
 condition_variable_t, 262
 CONDVAR_DECL, 262
 condition_variable, 580
 queue, 581
 condition_variable_t
 Condition Variables, 262
 CONDVAR_DECL
 Condition Variables, 262
 config
 ch_os_instance, 548
 Configuration, 31
 Core Memory Manager, 419
 __core_init, 422
 CH_CFG_MEMCORE_SIZE, 420
 ch_memcore, 429
 chCoreAlloc, 428
 chCoreAllocAligned, 427
 chCoreAllocAlignedI, 426
 chCoreAllocAlignedWithOffset, 421
 chCoreAllocAlignedWithOffsetI, 420
 chCoreAllocFromBase, 424
 chCoreAllocFromBaseI, 422
 chCoreAllocFromTop, 425
 chCoreAllocFromTopI, 423
 chCoreAllocI, 428
 chCoreGetStatusX, 426
 memgetfunc2_t, 421
 memgetfunc_t, 421
 core_id
 ch_os_instance, 547
 core_id_t
 OS Types and Structures, 63
 Counting Semaphores, 228
 __SEMAPHORE_DATA, 229
 chSemAddCounterI, 239
 chSemFastSignall, 243
 chSemFastWaitI, 243
 chSemGetCounterI, 244
 chSemObjectInit, 230
 chSemReset, 241
 chSemResetI, 242
 chSemResetWithMessage, 231
 chSemResetWithMessageI, 232
 chSemSignal, 237
 chSemSignall, 238
 chSemSignalWait, 240
 chSemWait, 233
 chSemWaitS, 234
 chSemWaitTimeout, 235
 chSemWaitTimeoutS, 236
 SEMAPHORE_DECL, 230
 semaphore_t, 230
 ctx
 ch_thread, 566
 cumulative
 time_measurement_t, 606
 currcore
 System Management, 114
 current
 ch_ready_list, 554
 Customer Information, 24
 CH_CUSTOMER_ID_CODE, 25
 CH_CUSTOMER_ID_STRING, 24
 CH_CUSTOMER_LICENSE_EOS_DATE, 25
 CH_CUSTOMER_LICENSE_VERSION_DATE, 25
 CH_CUSTOMER_LICENSE_VERSION_MONTH, 25
 CH_CUSTOMER_LICENSE_VERSION_YEAR, 25
 CH_LICENSE, 25
 dbg
 ch_os_instance, 548
 Debug, 324
 default_heap
 Memory Heaps, 436
 Delegate Threads, 391
 __ch_delegate_fn0, 392
 __ch_delegate_fn1, 393
 __ch_delegate_fn2, 393
 __ch_delegate_fn3, 393
 __ch_delegate_fn4, 395
 chDelegateCallDirect0, 397
 chDelegateCallDirect1, 398
 chDelegateCallDirect2, 399

chDelegateCallDirect3, 400
chDelegateCallDirect4, 401
chDelegateCallVeneer, 395
chDelegateDispatch, 396
chDelegateDispatchTimeout, 396
delegate_fn0_t, 392
delegate_fn1_t, 392
delegate_fn2_t, 392
delegate_fn3_t, 392
delegate_fn4_t, 392
delegate_veneer_t, 392
delegate_fn0_t
 Delegate Threads, 392
delegate_fn1_t
 Delegate Threads, 392
delegate_fn2_t
 Delegate Threads, 392
delegate_fn3_t
 Delegate Threads, 392
delegate_fn4_t
 Delegate Threads, 392
delegate_veneer_t
 Delegate Threads, 392
delta
 ch_delta_list, 517
dlist
 ch_virtual_timer, 573
 ch_virtual_timers_list, 575
dptr
 ch_oc_object, 545
dyn_buffer_t
 Dynamic Objects Factory, 495
dyn_element_t
 Dynamic Objects Factory, 495
dyn_list_t
 Dynamic Objects Factory, 495
dyn_mailbox_t
 Dynamic Objects Factory, 496
dyn_objects_fifo_t
 Dynamic Objects Factory, 496
dyn_pipe_t
 Dynamic Objects Factory, 496
dyn_semaphore_t
 Dynamic Objects Factory, 495
Dynamic Objects Factory, 490
 __factory_init, 496
 CH_CFG_FACTORY_GENERIC_BUFFERS, 493
 CH_CFG_FACTORY_MAILBOXES, 494
 CH_CFG_FACTORY_MAX_NAMES_LENGTH,
 493
 CH_CFG_FACTORY_OBJ_FIFOS, 494
 CH_CFG_FACTORY_OBJECTS_REGISTRY, 493
 CH_CFG_FACTORY_PIPES, 494, 495
 CH_CFG_FACTORY_SEMAPHORES, 493
ch_factory, 513
chFactoryCreateBuffer, 499
chFactoryCreateMailbox, 503
chFactoryCreateObjectsFIFO, 504
chFactoryCreatePipe, 506
chFactoryCreateSemaphore, 501
chFactoryDuplicateReference, 509
chFactoryFindBuffer, 500
chFactoryFindMailbox, 503
chFactoryFindObject, 498
chFactoryFindObjectByPointer, 498
chFactoryFindObjectsFIFO, 505
chFactoryFindPipe, 507
chFactoryFindSemaphore, 502
chFactoryGetBuffer, 510
chFactoryGetBufferSize, 510
chFactoryGetMailbox, 511
chFactoryGetObject, 509
chFactoryGetObjectsFIFO, 512
chFactoryGetPipe, 512
chFactoryGetSemaphore, 511
chFactoryRegisterObject, 497
chFactoryReleaseBuffer, 501
chFactoryReleaseMailbox, 504
chFactoryReleaseObject, 499
chFactoryReleaseObjectsFIFO, 506
chFactoryReleasePipe, 507
chFactoryReleaseSemaphore, 502
dyn_buffer_t, 495
dyn_element_t, 495
dyn_list_t, 495
dyn_mailbox_t, 496
dyn_objects_fifo_t, 496
dyn_pipe_t, 496
dyn_semaphore_t, 495
objects_factory_t, 496
registered_object_t, 495
Dynamic Threads, 311
 chThdCreateFromHeap, 311
 chThdCreateFromMemoryPool, 312
element
 ch_dyn_mailbox, 520
 ch_dyn_object, 522
 ch_dyn_objects_fifo, 523
 ch_dyn_pipe, 525
 ch_dyn_semaphore, 526
 ch_registered_static_object, 555
epending
 ch_thread, 570
Event Flags, 273
 __EVENTSOURCE_DATA, 275
 ALL_EVENTS, 275
 chEvtAddEvents, 281
 chEvtAddEventsI, 298
 chEvtBroadcast, 297
 chEvtBroadcastFlags, 287
 chEvtBroadcastFlagsI, 286
 chEvtBroadcastI, 297
 chEvtDispatch, 288
 chEvtGetAndClearEvents, 280
 chEvtGetAndClearEventsI, 279
 chEvtGetAndClearFlags, 283

chEvtGetAndClearFlagsI, 282
 chEvtGetEventsX, 298
 chEvtIsListeningI, 296
 chEvtObjectInit, 294
 chEvtRegister, 295
 chEvtRegisterMask, 295
 chEvtRegisterMaskWithFlags, 277
 chEvtRegisterMaskWithFlagsI, 276
 chEvtSignal, 285
 chEvtSignall, 284
 chEvtUnregister, 278
 chEvtWaitAll, 290
 chEvtWaitAllTimeout, 293
 chEvtWaitAny, 289
 chEvtWaitAnyTimeout, 292
 chEvtWaitOne, 288
 chEvtWaitOneTimeout, 291
 EVENT_MASK, 275
 event_source_t, 276
 EVENTSOURCE DECL, 276
 evhandler_t, 276
 event_listener, 581
 events, 582
 flags, 582
 listener, 582
 next, 582
 wflags, 583
 EVENT_MASK
 Event Flags, 275
 event_source, 583
 next, 584
 event_source_t
 Event Flags, 276
 eventflags_t
 OS Types and Structures, 63
 eventid_t
 OS Types and Structures, 62
 eventmask_t
 OS Types and Structures, 63
 events
 event_listener, 582
 EVENTSOURCE DECL
 Event Flags, 276
 evhandler_t
 Event Flags, 276
 ewmask
 ch_thread, 569
 exitcode
 ch_thread, 568
 FALSE
 Version Numbers and Identification, 30
 fifo
 ch_dyn_objects_fifo, 523
 fifo_list
 ch_objects_factory, 537
 firstprio
 Scheduler, 97
 flags
 ch_thread, 567
 event_listener, 582
 free
 ch_jobs_queue, 528
 ch_objects_fifo, 539
 func
 ch_virtual_timer, 573
 funcp
 thread_descriptor_t, 604
 guarded_memory_pool_t, 585
 pool, 586
 sem, 586
 GUARDEDMEMORYPOOL DECL
 Memory Pools, 440
 halt
 trace_event_t, 614
 hash_get_s
 Objects Caches, 482
 hash_next
 ch_oc_hash_header, 540
 ch_oc_lru_header, 542
 ch_oc_object, 544
 hash_prev
 ch_oc_hash_header, 541
 ch_oc_lru_header, 542
 ch_oc_object, 544
 hashn
 ch_objects_cache, 533
 hashp
 ch_objects_cache, 533
 hdr
 ch_thread, 565
 header
 memory_heap, 595
 heap
 heap_header, 587
 heap_header, 586
 heap, 587
 next, 587
 pages, 587
 size, 587
 heap_header_t
 Memory Heaps, 432
 HIGHPRIO
 Scheduler, 93
 identifier
 chdebug_t, 577
 IDLEPRIO
 Scheduler, 92
 idlethread_base
 ch_os_instance_config, 550
 idlethread_end
 ch_os_instance_config, 550
 instance
 thread_descriptor_t, 604
 instances

ch_system, 561
isr
 trace_event_t, 613
isr_cnt
 ch_system_debug, 562

job_descriptor_t
 Jobs Queues, 404
job_function_t
 Jobs Queues, 404
jobarg
 ch_job_descriptor, 527
jobfunc
 ch_job_descriptor, 527
Jobs Queues, 403
 chJobDispatch, 415
 chJobDispatchTimeout, 416
 chJobGet, 405
 chJobGetl, 406
 chJobGetTimeout, 408
 chJobGetTimeoutS, 407
 chJobObjectInit, 404
 chJobPost, 411
 chJobPostAhead, 414
 chJobPostAheadl, 412
 chJobPostAheadS, 413
 chJobPostl, 409
 chJobPostS, 410
 job_descriptor_t, 404
 job_function_t, 404
 jobs_queue_t, 404
 MSG_JOB_NULL, 404
jobs_queue_t
 Jobs Queues, 404

kernel_stats
 ch_os_instance, 548
kernel_stats_t, 588
 m_crit_isr, 589
 m_crit_thd, 589
 n_ctxswc, 589
 n_irq, 589

last
 time_measurement_t, 605
laststamp
 ch_virtual_timers_list, 575
lasttime
 ch_virtual_timers_list, 575
License Settings, 26
 CH_LICENSE_FEATURES, 27
 CH_LICENSE_ID_CODE, 27
 CH_LICENSE_ID_STRING, 27
 CH_LICENSE_MAX_DEPLOY, 27
 CH_LICENSE_MODIFIABLE_CODE, 27
 CH_LICENSE_TYPE_STRING, 26
likely
 OS Types and Structures, 60
list
 ch_thread, 565
listener
 event_listener, 582
Lists and Queues, 74
 __CH_QUEUE_DATA, 75
 ch_delta_list_t, 76
 ch_dlist_dequeue, 88
 ch_dlist_init, 84
 ch_dlist_insert, 87
 ch_dlist_insert_after, 86
 ch_dlist_insert_before, 86
 ch_dlist_isempty, 84
 ch_dlist_isfirst, 85
 ch_dlist_islast, 85
 ch_dlist_notempty, 85
 ch_dlist_remove_first, 87
 ch_list_init, 77
 ch_list_isempty, 77
 ch_list_link, 78
 ch_list_notempty, 77
 ch_list_t, 76
 ch_list_unlink, 78
 ch_pqueue_init, 82
 ch_pqueue_insert_ahead, 83
 ch_pqueue_insert_behind, 83
 ch_pqueue_remove_highest, 82
 ch_priority_queue_t, 76
 CH_QUEUE_DECL, 76
 ch_queue_dequeue, 81
 ch_queue_fifo_remove, 80
 ch_queue_init, 79
 ch_queue_insert, 80
 ch_queue_isempty, 79
 ch_queue_lifo_remove, 81
 ch_queue_notempty, 80
 ch_queue_t, 76
lock_cnt
 ch_system_debug, 563
LOWPRIO
 Scheduler, 92
lru
 ch_objects_cache, 534
lru_get_last_s
 Objects Caches, 483
lru_next
 ch_oc_lru_header, 542
 ch_oc_object, 544
lru_prev
 ch_oc_lru_header, 542
 ch_oc_object, 544
lru_sem
 ch_objects_cache, 534

m_crit_isr
 kernel_stats_t, 589
m_crit_thd
 kernel_stats_t, 589
MAILBOX_DECL
 Mailboxes, 367

mailbox_t, 590
 buffer, 591
 cnt, 592
 qr, 592
 qw, 592
 rptr, 591
 reset, 592
 top, 591
 wrptr, 591
Mailboxes, 365
 __MAILBOX_DATA, 366
chMBFetchl, 378
chMBFetchTimeout, 376
chMBFetchTimeoutS, 377
chMBGetFreeCountl, 380
chMBGetSizeI, 379
chMBGetUsedCountl, 380
chMBOBJECTInit, 367
chMBPeekl, 381
chMBPostAheadl, 375
chMBPostAheadTimeout, 373
chMBPostAheadTimeoutS, 374
chMBPostl, 372
chMBPostTimeout, 370
chMBPostTimeoutS, 371
chMBReset, 368
chMBResetl, 369
chMBResumeX, 382
MAILBOX_DECL, 367
mainthread
 ch_os_instance, 548
mainthread_base
 ch_os_instance_config, 549
mainthread_end
 ch_os_instance_config, 550
mask
 ch_rfcu, 557
mbx
 ch_dyn_mailbox, 521
 ch_jobs_queue, 529
 ch_objects_fifo, 539
mbx_list
 ch_objects_factory, 537
MEM_ALIGN_MASK
 Memory Alignment, 136
MEM_ALIGN_NEXT
 Memory Alignment, 137
MEM_ALIGN_PREV
 Memory Alignment, 136
MEM_IS_ALIGNED
 Memory Alignment, 137
MEM_IS_VALID_ALIGNMENT
 Memory Alignment, 137
memcore_t, 593
 basemem, 593
 topmem, 593
memgetfunc2_t
 Core Memory Manager, 421
memgetfunc_t
 Core Memory Manager, 421
Memory Alignment, 136
 MEM_ALIGN_MASK, 136
 MEM_ALIGN_NEXT, 137
 MEM_ALIGN_PREV, 136
 MEM_IS_ALIGNED, 137
 MEM_IS_VALID_ALIGNMENT, 137
Memory Heaps, 430
 __heap_init, 432
 CH_HEAP_ALIGNMENT, 431
 CH_HEAP_AREA, 431
 chHeapAlloc, 434
 chHeapAllocAligned, 433
 chHeapFree, 433
 chHeapGetSize, 435
 chHeapObjectInit, 432
 chHeapStatus, 434
 default_heap, 436
 heap_header_t, 432
 memory_heap_t, 432
Memory Management, 418
Memory Pools, 437
 __GUARDEDMEMORYPOOL_DATA, 439
 __MEMORYPOOL_DATA, 438
 chGuardedPoolAdd, 458
 chGuardedPoolAddl, 459
 chGuardedPoolAddS, 460
 chGuardedPoolAllocI, 455
 chGuardedPoolAllocTimeout, 449
 chGuardedPoolAllocTimeoutS, 448
 chGuardedPoolFree, 450
 chGuardedPoolFreeI, 456
 chGuardedPoolFreeS, 457
 chGuardedPoolGetCounterI, 454
 chGuardedPoolLoadArray, 447
 chGuardedPoolObjectInit, 454
 chGuardedPoolObjectInitAligned, 446
 chPoolAdd, 452
 chPoolAddl, 453
 chPoolAlloc, 443
 chPoolAllocI, 442
 chPoolFree, 445
 chPoolFreeI, 444
 chPoolLoadArray, 441
 chPoolObjectInit, 451
 chPoolObjectInitAligned, 440
GUARDEDMEMORYPOOL_DECL, 440
MEMORYPOOL_DECL, 439
memory_heap, 594
 header, 595
 mtx, 595
 provider, 595
memory_heap_t
 Memory Heaps, 432
memory_pool_t, 596
 align, 597
 next, 596

object_size, 597
provider, 597
MEMORYPOOL_DECL
 Memory Pools, 439
mpool
 ch_thread, 571
MS2RTC
 System Management, 117
msg
 trace_event_t, 613
MSG_JOB_NULL
 Jobs Queues, 404
MSG_OK
 Scheduler, 92
MSG_RESET
 Scheduler, 92
msg_t
 OS Types and Structures, 62
MSG_TIMEOUT
 Scheduler, 92
msgqueue
 ch_thread, 570
mtx
 ch_objects_factory, 536
 memory_heap, 595
mtxlist
 ch_thread, 570
MUTEX_DECL
 Mutexes, 249
mutex_t
 Mutexes, 249
Mutexes, 246
 __MUTEX_DATA, 247
 chMtxGetNextMutexX, 260
 chMtxGetOwnerI, 259
 chMtxLock, 250
 chMtxLockS, 251
 chMtxObjectInit, 249
 chMtxQueueNotEmptyS, 258
 chMtxTryLock, 252
 chMtxTryLockS, 253
 chMtxUnlock, 254
 chMtxUnlockAll, 257
 chMtxUnlockAllS, 256
 chMtxUnlockS, 255
 MUTEX_DECL, 249
 mutex_t, 249

n
 time_measurement_t, 606

n_ctxswc
 kernel_stats_t, 589

n_irq
 kernel_stats_t, 589

name
 ch_os_instance_config, 549
 ch_thread, 566
 thread_descriptor_t, 603
 trace_event_t, 613

next
 ch_delta_list, 517
 ch_dyn_element, 518
 ch_list, 529
 ch_mutex, 531
 ch_priority_queue, 551
 ch_queue, 552
 event_listener, 582
 event_source, 584
 heap_header, 587
 memory_pool_t, 596
 pool_header, 602

NOPRIO
 Scheduler, 92

NORMALPRIO
 Scheduler, 93

ntp
 trace_event_t, 612

obj_flags
 ch_oc_object, 545

obj_group
 ch_oc_object, 544

obj_key
 ch_oc_object, 545

obj_list
 ch_objects_factory, 536

obj_pool
 ch_objects_factory, 536

obj_sem
 ch_oc_object, 545

object_size
 memory_pool_t, 597

Objects Caches, 480
 chCacheGetObject, 485
 chCacheObjectInit, 484
 chCacheReadObject, 486
 chCacheReleaseObject, 488
 chCacheReleaseObjectI, 486
 chCacheWriteObject, 487
 hash_get_s, 482
 lru_get_last_s, 483
 objects_cache_t, 482

oc_flags_t, 481

oc_hash_header_t, 481

oc_lru_header_t, 481

oc_object_t, 481

oc_readf_t, 482

oc_writef_t, 482

Objects FIFOs, 463
 chFifoObjectInit, 465
 chFifoObjectInitAligned, 464
 chFifoReceiveObjectI, 476
 chFifoReceiveObjectTimeout, 478
 chFifoReceiveObjectTimeoutS, 477
 chFifoReturnObject, 470
 chFifoReturnObjectI, 469
 chFifoReturnObjectS, 469
 chFifoSendObject, 473

chFifoSendObjectAhead, 476
 chFifoSendObjectAheadI, 474
 chFifoSendObjectAheadS, 475
 chFifoSendObjectI, 471
 chFifoSendObjectS, 472
 chFifoTakeObjectI, 466
 chFifoTakeObjectTimeout, 468
 chFifoTakeObjectTimeoutS, 467
 objects_fifo_t, 464
objects_cache_t
 Objects Caches, 482
objects_factory_t
 Dynamic Objects Factory, 496
objects_fifo_t
 Objects FIFOs, 464
objn
 ch_objects_cache, 533
objp
 ch_registered_static_object, 555
objsz
 ch_objects_cache, 534
objvp
 ch_objects_cache, 534
oc_flags_t
 Objects Caches, 481
oc_hash_header_t
 Objects Caches, 481
oc_lru_header_t
 Objects Caches, 481
oc_object_t
 Objects Caches, 481
oc_readf_t
 Objects Caches, 482
oc_writef_t
 Objects Caches, 482
off_ctx
 chdebug_t, 578
off_flags
 chdebug_t, 579
off_name
 chdebug_t, 579
off_newer
 chdebug_t, 579
off_older
 chdebug_t, 579
off_preempt
 chdebug_t, 580
off_prio
 chdebug_t, 578
off_refs
 chdebug_t, 580
off_state
 chdebug_t, 579
off_stklimit
 chdebug_t, 579
off_time
 chdebug_t, 580
offset
 tm_calibration_t, 607
Options, 32
 CH_CFG_CONTEXT_SWITCH_HOOK, 51
 CH_CFG_FACTORY_GENERIC_BUFFERS, 45
 CH_CFG_FACTORY_MAILBOXES, 46
 CH_CFG_FACTORY_MAX_NAMES_LENGTH, 45
 CH_CFG_FACTORY_OBJ_FIFOS, 46
 CH_CFG_FACTORY_OBJECTS_REGISTRY, 45
 CH_CFG_FACTORY_PIPES, 46
 CH_CFG_FACTORY_SEMAPHORES, 45
 CH_CFG_IDLE_ENTER_HOOK, 51
 CH_CFG_IDLE_LEAVE_HOOK, 52
 CH_CFG_IDLE_LOOP_HOOK, 52
 CH_CFG_INTERVALS_SIZE, 36
 CH_CFG_IRQ_EPILOGUE_HOOK, 51
 CH_CFG_IRQ_PROLOGUE_HOOK, 51
 CH_CFG_MEMCORE_SIZE, 42
 CH_CFG_NO_IDLE_THREAD, 37
 CH_CFG_OPTIMIZE_SPEED, 37
 CH_CFG_OS_INSTANCE_EXTRA_FIELDS, 49
 CH_CFG_OS_INSTANCE_INIT_HOOK, 49
 CH_CFG_RUNTIME_FAULTS_HOOK, 53
 CH_CFG_SMP_MODE, 35
 CH_CFG_ST_FREQUENCY, 36
 CH_CFG_ST_RESOLUTION, 35
 CH_CFG_ST_TIMedelta, 36
 CH_CFG_SYSTEM_EXTRA_FIELDS, 49
 CH_CFG_SYSTEM_HALT_HOOK, 53
 CH_CFG_SYSTEM_INIT_HOOK, 49
 CH_CFG_SYSTEM_TICK_HOOK, 52
 CH_CFG_THREAD_EXIT_HOOK, 50
 CH_CFG_THREAD_EXTRA_FIELDS, 50
 CH_CFG_THREAD_INIT_HOOK, 50
 CH_CFG_TIME_QUANTUM, 37
 CH_CFG_TIME_TYPES_SIZE, 36
 CH_CFG_TRACE_HOOK, 53
 CH_CFG_USE_CONDVARs, 39
 CH_CFG_USE_CONDVARs_TIMEOUT, 40
 CH_CFG_USE_DELEGATES, 44
 CH_CFG_USE_DYNAMIC, 41
 CH_CFG_USE_EVENTS, 40
 CH_CFG_USE_EVENTS_TIMEOUT, 40
 CH_CFG_USE_FACTORY, 45
 CH_CFG_USE_HEAP, 43
 CH_CFG_USE_JOBS, 44
 CH_CFG_USE_MAILBOXES, 42
 CH_CFG_USE_MEMCORE, 42
 CH_CFG_USE_MEMPOOLS, 43
 CH_CFG_USE_MESSAGES, 41
 CH_CFG_USE_MESSAGES_PRIORITY, 41
 CH_CFG_USE_MUTEXES, 39
 CH_CFG_USE_MUTEXES_RECURSIVE, 39
 CH_CFG_USE_OBJ_CACHES, 44
 CH_CFG_USE_OBJ_FIFOS, 43
 CH_CFG_USE_PIPES, 44
 CH_CFG_USE_REGISTRY, 38
 CH_CFG_USE_SEMAPHORES, 38
 CH_CFG_USE_SEMAPHORES_PRIORITY, 39

CH_CFG_USE_TIMESTAMP, 38
CH_CFG_USE_TM, 37
CH_CFG_USE_WAITEXIT, 38
CH_DBG_ENABLE_ASSERTS, 47
CH_DBG_ENABLE_CHECKS, 47
CH_DBG_ENABLE_STACK_CHECK, 48
CH_DBG_FILL_THREADS, 48
CH_DBG_STATISTICS, 46
CH_DBG_SYSTEM_STATE_CHECK, 46
CH_DBG_THREADS_PROFILING, 48
CH_DBG_TRACE_BUFFER_SIZE, 47
CH_DBG_TRACE_MASK, 47
OS Instances, 68
 __idle_thread, 69
 __instance_get_currthread, 68
 __instance_set_currthread, 68
 chInstanceObjectInit, 69
OS Library, 349
OS Types and Structures, 58
 __CH_OFFSETOF, 60
 __CH_STRINGIFY, 59
 __CH_USED, 60
 ch_system_t, 65
 chSysHalt, 66
 cnt_t, 63
 core_id_t, 63
 eventflags_t, 63
 eventid_t, 62
 eventmask_t, 63
 likely, 60
 msg_t, 62
 os_instance_config_t, 65
 os_instance_t, 64
 ready_list_t, 65
 registry_t, 65
 rtcnt_t, 61
 rttime_t, 61
 stkalign_t, 61
 syssts_t, 61
 system_state_t, 66
 thread_reference_t, 65
 thread_t, 63
 threads_queue_t, 65
 tmode_t, 62
 tprio_t, 62
 trefs_t, 62
 tslices_t, 62
 tstate_t, 62
 ucnt_t, 63
 unlikely, 61
 virtual_timer_t, 64
 virtual_timers_list_t, 64
 vfunc_t, 64
os_instance_config_t
 OS Types and Structures, 65
os_instance_t
 OS Types and Structures, 64
owner
 ch_mutex, 531
 ch_thread, 566
pages
 heap_header, 587
panic_msg
 ch_system_debug, 562
par
 ch_virtual_timer, 573
pipe
 ch_dyn_pipe, 525
PIPE DECL
 Pipes, 384
pipe_list
 ch_objects_factory, 537
pipe_read
 Pipes, 385
pipe_t, 597
 buffer, 599
 cmtx, 600
 cnt, 600
 rdptr, 600
 reset, 600
 rmtx, 601
 rtr, 600
 top, 599
 wmtx, 601
 wrptr, 599
 wtr, 600
pipe_write
 Pipes, 384
Pipes, 383
 __PIPE_DATA, 383
 chPipeGetFreeCount, 389
 chPipeGetSize, 388
 chPipeGetUsedCount, 388
 chPipeObjectInit, 386
 chPipeReadTimeout, 387
 chPipeReset, 386
 chPipeResume, 389
 chPipeWriteTimeout, 386
 PIPE DECL, 384
 pipe_read, 385
 pipe_write, 384
pool
 guarded_memory_pool_t, 586
pool_header, 601
 next, 602
Port Interface, 57
pqueue
 ch_ready_list, 554
 ch_thread, 565
prev
 ch_delta_list, 517
 ch_priority_queue, 551
 ch_queue, 552
prio
 ch_priority_queue, 551
 thread_descriptor_t, 603

provider
 memory_heap, 595
 memory_pool_t, 597
 ptr
 trace_buffer_t, 609
 ptrsize
 chdebug_t, 578
 qr
 mailbox_t, 592
 queue
 ch_mutex, 531
 ch_registry, 556
 ch_semaphore, 559
 ch_thread, 565
 ch_threads_queue, 572
 condition_variable, 581
 qw
 mailbox_t, 592
 rdptr
 mailbox_t, 591
 pipe_t, 600
 rdy
 trace_event_t, 613
 rdymsg
 ch_thread, 567
 readf
 ch_objects_cache, 534
 ready_list_t
 OS Types and Structures, 65
 realprio
 ch_thread, 570
 reason
 trace_event_t, 614
 refs
 ch_dyn_element, 518
 ch_thread, 567
 REG_HEADER
 Registry, 315
 REG_INSERT
 Registry, 316
 REG_REMOVE
 Registry, 315
 registered_object_t
 Dynamic Objects Factory, 495
 Registry, 314
 __reg_object_init, 321
 chRegFindThreadByName, 318
 chRegFindThreadByPointer, 319
 chRegFindThreadByWorkingArea, 320
 chRegFirstThread, 316
 chRegGetThreadNameX, 322
 chRegNextThread, 317
 chRegSetThreadName, 321
 chRegSetThreadNameX, 322
 REG_HEADER, 315
 REG_INSERT, 316
 REG_REMOVE, 315
 registry_t
 OS Types and Structures, 65
 reglist
 ch_os_instance, 547
 ch_system, 561
 Release and Licensing, 21
 Release Information, 22
 __CHIBIOS__, 22
 CH_VERSION, 22
 CH_VERSION_DATE, 23
 CH_VERSION_MONTH, 23
 CH_VERSION_NICKNAME, 23
 CH_VERSION_PATCH, 23
 CH_VERSION_STABLE, 22
 CH_VERSION_YEAR, 23
 reload
 ch_virtual_timer, 573
 reset
 mailbox_t, 592
 pipe_t, 600
 Restrictions, 55
 rfcu
 ch_os_instance, 547
 ch_system, 561
 rfcu_mask_t
 Runtime Faults Collection Unit, 72
 rfcu_t
 Runtime Faults Collection Unit, 72
 rlist
 ch_os_instance, 547
 rmtx
 pipe_t, 601
 rqueue
 ch_thread, 566
 RT Kernel, 28
 RTC2MS
 System Management, 119
 RTC2S
 System Management, 118
 RTC2US
 System Management, 119
 rtcnt_t
 OS Types and Structures, 61
 rtr
 pipe_t, 600
 rtstamp
 trace_event_t, 612
 rttime_t
 OS Types and Structures, 61
 Runtime Faults Collection Unit, 71
 __rfcu_object_init, 73
 CH_RFCU_ALLFAULTS, 71
 chRFCUCollectFaultsI, 72
 chRFCUGetAndClearFaultsI, 72
 rfcu_mask_t, 72
 rfcu_t, 72
 S2RTC
 System Management, 116

Scheduler, 89
 __sch_get_currthread, 97
 __sch_ready_ahead, 99
 __sch_ready_behind, 97
 __sch_reschedule_ahead, 100
 __sch_reschedule_behind, 100
 CH_FLAG_MODE_HEAP, 96
 CH_FLAG_MODE_MASK, 96
 CH_FLAG_MODE_MPOOL, 96
 CH_FLAG_MODE_STATIC, 96
 CH_FLAG_TERMINATE, 97
 ch_sch_prio_insert, 101
 CH_STATE_CURRENT, 93
 CH_STATE_FINAL, 95
 CH_STATE_NAMES, 96
 CH_STATE_QUEUED, 94
 CH_STATE_READY, 93
 CH_STATE_SLEEPING, 94
 CH_STATE SNDMSG, 95
 CH_STATE SNDMSGQ, 95
 CH_STATE_SUSPENDED, 93
 CH_STATE_WTANDEV, 95
 CH_STATE_WTCOND, 94
 CH_STATE_WTEXIT, 94
 CH_STATE_WTMSG, 95
 CH_STATE_WTMTX, 94
 CH_STATE_WTOREVT, 95
 CH_STATE_WTSEM, 94
 CH_STATE_WTSTART, 93
 chSchDoPreemption, 107
 chSchDoYieldS, 108
 chSchGoSleepS, 103
 chSchGoSleepTimeoutS, 104
 chSchIsPreemptionRequired, 106
 chSchPreemption, 108
 chSchReadyI, 102
 chSchRescheduleS, 106
 chSchSelectFirstI, 109
 chSchWakeups, 105
 firstprio, 97
 HIGHPRIO, 93
 IDLEPRIO, 92
 LOWPRIO, 92
 MSG_OK, 92
 MSG_RESET, 92
 MSG_TIMEOUT, 92
 NOPRIO, 92
 NORMALPRIO, 93

sem
 ch_dyn_semaphore, 526
 guarded_memory_pool_t, 586

sem_list
 ch_objects_factory, 537

sem_pool
 ch_objects_factory, 537

SEMAPHORE DECL
 Counting Semaphores, 230

semaphore_t

 Counting Semaphores, 230

sentmsg
 ch_thread, 568

size
 chdebug_t, 577
 heap_header, 587
 trace_buffer_t, 609

state
 ch_system, 560
 ch_thread, 566
 trace_event_t, 612

Statistics, 345
 __stats_ctxswc, 345
 __stats_increase_irq, 345
 __stats_object_init, 347
 __stats_start_measure_crit_isr, 347
 __stats_start_measure_crit_thd, 346
 __stats_stop_measure_crit_isr, 347
 __stats_stop_measure_crit_thd, 346

stats
 ch_thread, 571

stkalign_t
 OS Types and Structures, 61

suspended
 trace_buffer_t, 609

sw
 trace_event_t, 613

Synchronization, 227, 353

Synchronous Messages, 300
 chMsgGet, 309
 chMsgIsPendingI, 308
 chMsgPoll, 307
 chMsgPollS, 303
 chMsgRelease, 304
 chMsgReleaseS, 310
 chMsgSend, 301
 chMsgWait, 305
 chMsgWaitS, 301
 chMsgWaitTimeout, 306
 chMsgWaitTimeoutS, 302

sysinterval_t
 Time and Intervals, 145

syssts_t
 OS Types and Structures, 61

System, 56

System Management, 111
 ch0, 134
 ch1, 135
 ch_core0_cfg, 135
 ch_core1_cfg, 135
 CH_FAST_IRQ_HANDLER, 116
 CH_IRQ_EPILOGUE, 115
 CH_IRQ_HANDLER, 116
 CH_IRQ_IS_VALID_KERNEL_PRIORITY, 114
 CH_IRQ_IS_VALID_PRIORITY, 114
 CH_IRQ_PROLOGUE, 115
 CH_SYS_CORE0_MEMORY, 113
 CH_SYS_CORE1_MEMORY, 113

ch_system, 134
 chSysDisable, 128
 chSysEnable, 129
 chSysGetIdleThreadX, 134
 chSysGetRealtimeCounterX, 120
 chSysGetStatusAndLockX, 125
 chSysHalt, 122
 chSysInit, 122
 chSysIntegrityCheckI, 123
 chSysIsCounterWithinX, 127
 chSysLock, 130
 chSysLockFromISR, 131
 chSysNotifyInstance, 133
 chSysPolledDelayX, 128
 chSysRestoreStatusX, 126
 chSysSuspend, 129
 chSysSwitch, 120
 chSysTimerHandlerI, 124
 chSysUnconditionalLock, 132
 chSysUnconditionalUnlock, 133
 chSysUnlock, 130
 chSysUnlockFromISR, 132
 chSysWaitSystemState, 121
 currcore, 114
 MS2RTC, 117
 RTC2MS, 119
 RTC2S, 118
 RTC2US, 119
 S2RTC, 116
 THD_WORKING_AREA, 121
 US2RTC, 118
 system_debug_t
 Checks and Assertions, 328
 system_state_t
 OS Types and Structures, 66
 systime
 ch_virtual_timers_list, 575
 systime_t
 Time and Intervals, 145
 systimestamp_t
 Time and Intervals, 145
 tfunc_t
 Threads, 189
 THD_DESCRIPTOR
 Threads, 185
 THD_DESCRIPTOR_AFFINITY
 Threads, 187
 THD_FUNCTION
 Threads, 185
 THD_WORKING_AREA
 System Management, 121
 Threads, 184
 THD_WORKING_AREA_BASE
 Threads, 184
 THD_WORKING_AREA_END
 Threads, 185
 THD_WORKING_AREA_SIZE
 Threads, 183
 thread_descriptor_t, 602
 arg, 604
 funcp, 604
 instance, 604
 name, 603
 prio, 603
 wbase, 603
 wend, 603
 thread_reference_t
 OS Types and Structures, 65
 thread_t
 OS Types and Structures, 63
 Threads, 180
 __THREADS_QUEUE_DATA, 183
 __thd_memfill, 190
 __thd_object_init, 189
 chThdAddRef, 197
 chThdCreate, 193
 chThdCreateI, 192
 chThdCreateStatic, 195
 chThdCreateSuspended, 191
 chThdCreateSuspendedI, 190
 chThdDequeueAllI, 215
 chThdDequeueNextI, 214
 chThdDoDequeueNextI, 221
 chThdEnqueueTimeoutS, 213
 chThdExit, 199
 chThdExitS, 200
 chThdGetPriorityX, 216
 chThdGetSelfX, 216
 chThdGetTicksX, 216
 chThdGetWorkingAreaX, 217
 chThdQueueIsEmptyI, 220
 chThdQueueObjectInit, 220
 chThdRelease, 198
 chThdResume, 212
 chThdResumel, 210
 chThdResumeS, 211
 chThdSetPriority, 202
 chThdShouldTerminateX, 218
 chThdSleep, 204
 chThdSleepMicroseconds, 188
 chThdSleepMilliseconds, 188
 chThdSleepS, 219
 chThdSleepSeconds, 187
 chThdSleepUntil, 205
 chThdSleepUntilWindowed, 206
 chThdStart, 196
 chThdStartI, 218
 chThdSuspendS, 208
 chThdSuspendTimeoutS, 209
 chThdTerminate, 203
 chThdTerminatedX, 217
 chThdWait, 201
 chThdYield, 207
 tfunc_t, 189
 THD_DESCRIPTOR, 185
 THD_DESCRIPTOR_AFFINITY, 187

THD_FUNCTION, 185
THD_WORKING_AREA, 184
THD_WORKING_AREA_BASE, 184
THD_WORKING_AREA_END, 185
THD_WORKING_AREA_SIZE, 183
THREADS_QUEUE_DECL, 183
THREADS_QUEUE_DECL
 Threads, 183
threads_queue_t
 OS Types and Structures, 65
threadsize
 chdebug_t, 578
ticks
 ch_thread, 567
time
 ch_thread, 567
 trace_event_t, 612
Time and Intervals, 139
 chTimeAddX, 149
 chTimeDiffX, 150
 chTimel2MS, 148
 chTimel2S, 148
 chTimel2US, 149
 chTimelIsInRangeX, 150
 chTimeMS2I, 147
 chTimeS2I, 146
 chTimeStampAddX, 151
 chTimeStampDiffX, 151
 chTimeStampIsInRangeX, 152
 chTimeUS2I, 147
 sysinterval_t, 145
 systime_t, 145
 systimestamp_t, 145
 time_conv_t, 146
TIME_I2MS, 143
TIME_I2S, 143
TIME_I2US, 144
TIME_IMMEDIATE, 140
TIME_INFINITE, 140
TIME_MAX_INTERVAL, 141
TIME_MAX_SYSTIME, 141
TIME_MS2I, 142
time_msecs_t, 146
TIME_S2I, 141
time_secs_t, 145
TIME_US2I, 142
time_usecs_t, 146
Time Measurement, 223
 __tm_calibration_object_init, 225
 chTMChainMeasurementToX, 225
 chTMOBJECTINIT, 223
 chTMStartMeasurementX, 224
 chTMStopMeasurementX, 224
 TM_CALIBRATION_LOOP, 223
time_conv_t
 Time and Intervals, 146
TIME_I2MS
 Time and Intervals, 143
TIME_I2S
 Time and Intervals, 143
TIME_I2US
 Time and Intervals, 144
TIME_IMMEDIATE
 Time and Intervals, 140
TIME_INFINITE
 Time and Intervals, 140
TIME_MAX_INTERVAL
 Time and Intervals, 141
TIME_MAX_SYSTIME
 Time and Intervals, 141
time_measurement_t, 604
 best, 605
 cumulative, 606
 last, 605
 n, 606
 worst, 605
TIME_MS2I
 Time and Intervals, 142
time_msecs_t
 Time and Intervals, 146
TIME_S2I
 Time and Intervals, 141
time_secs_t
 Time and Intervals, 145
TIME_US2I
 Time and Intervals, 142
time_usecs_t
 Time and Intervals, 146
timesize
 chdebug_t, 578
TM_CALIBRATION_LOOP
 Time Measurement, 223
tm_calibration_t, 606
 offset, 607
tmc
 ch_system, 561
tmode_t
 OS Types and Structures, 62
top
 mailbox_t, 591
 pipe_t, 599
topmem
 memcore_t, 593
tp
 trace_event_t, 613
tprio_t
 OS Types and Structures, 62
trace_buffer
 ch_os_instance, 548
trace_buffer_t, 607
 buffer, 609
 ptr, 609
 size, 609
 suspended, 609
trace_event_t, 610
 halt, 614

isr, 613
 msg, 613
 name, 613
 ntp, 612
 rdy, 613
 reason, 614
 rtstamp, 612
 state, 612
 sw, 613
 time, 612
 tp, 613
 type, 612
 up1, 614
 up2, 614
 user, 614
 wtobjp, 612
trace_next
 Tracing, 337
Tracing, 335
 __trace_halt, 339
 __trace_isr_enter, 338
 __trace_isr_leave, 339
 __trace_object_init, 337
 __trace_ready, 337
 __trace_switch, 338
 CH_DBG_TRACE_BUFFER_SIZE, 336
 CH_DBG_TRACE_MASK, 336
 chTraceResume, 343
 chTraceResumel, 343
 chTraceSuspend, 342
 chTraceSuspendl, 341
 chTraceWrite, 340
 chTraceWritel, 339
 trace_next, 337
trefs_t
 OS Types and Structures, 62
TRUE
 Version Numbers and Identification, 30
tslices_t
 OS Types and Structures, 62
tstate_t
 OS Types and Structures, 62
type
 trace_event_t, 612
u
 ch_thread, 569
ucnt_t
 OS Types and Structures, 63
unlikely
 OS Types and Structures, 61
up1
 trace_event_t, 614
up2
 trace_event_t, 614
US2RTC
 System Management, 118
user
 trace_event_t, 614
version
 chdebug_t, 578
Version Numbers and Identification, 29, 350
 __CHIBIOS_OSLIB__, 350
 __CHIBIOS_RT__, 29
 __oslib_init, 351
 CH_KERNEL_MAJOR, 30
 CH_KERNEL_MINOR, 30
 CH_KERNEL_PATCH, 30
 CH_KERNEL_STABLE, 29
 CH_KERNEL_VERSION, 30
 CH_OSLIB_MAJOR, 351
 CH_OSLIB_MINOR, 351
 CH_OSLIB_PATCH, 351
 CH_OSLIB_STABLE, 350
 CH_OSLIB_VERSION, 350
 FALSE, 30
 TRUE, 30
Virtual Timers, 153
 __vt_object_init, 178
 chVTDResetl, 157
 chVTDSetContinuousl, 156
 chVTDSetl, 155
 chVTDoTickl, 159
 chVTGetReloadIntervalX, 175
 chVTGetRemainingInterval, 158
 chVTGetSystemTime, 162
 chVTGetSystemTimeX, 162
 chVTGetTimersStatel, 166
 chVTGetTimeStamp, 176
 chVTGetTimeStampl, 160
 chVTIsArmed, 168
 chVTIsArmedl, 167
 chVTIsSystemTimeWithin, 165
 chVTIsSystemTimeWithinX, 164
 chVTOBJECTInit, 161
 chVTReset, 170
 chVTResetl, 169
 chVTRestartTimeStamp, 177
 chVTRestartTimeStampl, 161
 chVTSet, 172
 chVTSetContinuous, 174
 chVTSetContinuousl, 173
 chVTSetl, 171
 chVTSetReloadIntervalX, 175
 chVTTIMEElapsedSinceX, 163
 vt_enqueue, 154
 vt_insert_first, 154
virtual_timer_t
 OS Types and Structures, 64
virtual_timers_list_t
 OS Types and Structures, 64
vt_enqueue
 Virtual Timers, 154
vt_insert_first
 Virtual Timers, 154
vtfunc_t
 OS Types and Structures, 64

vtlist
 ch_os_instance, [547](#)

wabase
 ch_thread, [566](#)

waiting
 ch_thread, [570](#)

wbase
 thread_descriptor_t, [603](#)

wend
 thread_descriptor_t, [603](#)

wflags
 event_listener, [583](#)

wmtx
 pipe_t, [601](#)

worst
 time_measurement_t, [605](#)

writef
 ch_objects_cache, [535](#)

wrptr
 mailbox_t, [591](#)
 pipe_t, [599](#)

wtmtpx
 ch_thread, [569](#)

wtobjp
 ch_thread, [568](#)
 trace_event_t, [612](#)

wtr
 pipe_t, [600](#)

wtsemp
 ch_thread, [569](#)

wttrp
 ch_thread, [568](#)

zero
 chdebug_t, [577](#)