

ChibiOS/NIL

3.0.0

Reference Manual

Tue May 1 2018 09:42:48

Contents

1	ChibiOS/NIL	1
1.1	Copyright	1
1.2	Introduction	1
1.3	Related Documents	1
2	Module Index	3
2.1	Modules	3
3	Hierarchical Index	5
3.1	Class Hierarchy	5
4	Data Structure Index	7
4.1	Data Structures	7
5	File Index	9
5.1	File List	9
6	Module Documentation	11
6.1	NIL Kernel	11
6.1.1	Detailed Description	11
6.2	Configuration	12
6.2.1	Detailed Description	12
6.2.2	Macro Definition Documentation	13
6.2.2.1	CH_CFG_NUM_THREADS	13
6.2.2.2	CH_CFG_ST_RESOLUTION	13
6.2.2.3	CH_CFG_ST_FREQUENCY	14
6.2.2.4	CH_CFG_ST_TIMEDELTA	14
6.2.2.5	CH_CFG_USE_SEMAPHORES	14
6.2.2.6	CH_CFG_USE_MUTEXES	14
6.2.2.7	CH_CFG_USE_EVENTS	14
6.2.2.8	CH_CFG_USE_MAILBOXES	14
6.2.2.9	CH_CFG_USE_MEMCORE	15
6.2.2.10	CH_CFG_USE_HEAP	15

6.2.2.11	CH_CFG_USE_MEMPOOLS	15
6.2.2.12	CH_CFG_USE_OBJ_FIFOS	15
6.2.2.13	CH_CFG_MEMCORE_SIZE	15
6.2.2.14	CH_CFG_USE_FACTORY	16
6.2.2.15	CH_CFG_FACTORY_MAX_NAMES_LENGTH	16
6.2.2.16	CH_CFG_FACTORY_OBJECTS_REGISTRY	16
6.2.2.17	CH_CFG_FACTORY_GENERIC_BUFFERS	16
6.2.2.18	CH_CFG_FACTORY_SEMAPHORES	16
6.2.2.19	CH_CFG_FACTORY_MAILBOXES	16
6.2.2.20	CH_CFG_FACTORY_OBJ_FIFOS	16
6.2.2.21	CH_DBG_STATISTICS	16
6.2.2.22	CH_DBG_SYSTEM_STATE_CHECK	16
6.2.2.23	CH_DBG_ENABLE_CHECKS	17
6.2.2.24	CH_DBG_ENABLE_ASSERTS	17
6.2.2.25	CH_DBG_ENABLE_STACK_CHECK	17
6.2.2.26	CH_CFG_SYSTEM_INIT_HOOK	17
6.2.2.27	CH_CFG_THREAD_EXT_FIELDS	17
6.2.2.28	CH_CFG_THREAD_EXT_INIT_HOOK	17
6.2.2.29	CH_CFG_IDLE_ENTER_HOOK	18
6.2.2.30	CH_CFG_IDLE_LEAVE_HOOK	18
6.2.2.31	CH_CFG_SYSTEM_HALT_HOOK	18
6.3	API	19
6.3.1	Detailed Description	19
6.3.2	Macro Definition Documentation	26
6.3.2.1	_CHIBIOS_NIL	26
6.3.2.2	CH_KERNEL_STABLE	26
6.3.2.3	CH_KERNEL_VERSION	26
6.3.2.4	CH_KERNEL_MAJOR	26
6.3.2.5	CH_KERNEL_MINOR	26
6.3.2.6	CH_KERNEL_PATCH	26
6.3.2.7	MSG_OK	26
6.3.2.8	MSG_TIMEOUT	26
6.3.2.9	MSG_RESET	26
6.3.2.10	TIME_IMMEDIATE	27
6.3.2.11	TIME_INFINITE	27
6.3.2.12	TIME_MAX_INTERVAL	27
6.3.2.13	TIME_MAX_SYSTIME	27
6.3.2.14	NIL_STATE_READY	27
6.3.2.15	NIL_STATE_SLEEPING	27
6.3.2.16	NIL_STATE_SUSP	27

6.3.2.17	NIL_STATE_WTQUEUE	27
6.3.2.18	NIL_STATE_WTOREVT	27
6.3.2.19	ALL_EVENTS	27
6.3.2.20	EVENT_MASK	27
6.3.2.21	CH_CFG_USE_FACTORY	28
6.3.2.22	CH_CFG_FACTORY_MAX_NAMES_LENGTH	28
6.3.2.23	CH_CFG_FACTORY_OBJECTS_REGISTRY	28
6.3.2.24	CH_CFG_FACTORY_GENERIC_BUFFERS	28
6.3.2.25	CH_CFG_FACTORY_SEMAPHORES	28
6.3.2.26	CH_CFG_FACTORY_MAILBOXES	28
6.3.2.27	CH_CFG_FACTORY_OBJ_FIFOS	28
6.3.2.28	THD_IDLE_BASE	28
6.3.2.29	__CH_STRINGIFY	28
6.3.2.30	THD_TABLE_BEGIN	28
6.3.2.31	THD_TABLE_ENTRY	29
6.3.2.32	THD_TABLE_END	29
6.3.2.33	MEM_ALIGN_MASK	29
6.3.2.34	MEM_ALIGN_PREV	29
6.3.2.35	MEM_ALIGN_NEXT	29
6.3.2.36	MEM_IS_ALIGNED	30
6.3.2.37	MEM_IS_VALID_ALIGNMENT	30
6.3.2.38	THD_WORKING_AREA_SIZE	30
6.3.2.39	THD_WORKING_AREA	30
6.3.2.40	THD_FUNCTION	31
6.3.2.41	CH_IRQ_IS_VALID_PRIORITY	31
6.3.2.42	CH_IRQ_IS_VALID_KERNEL_PRIORITY	31
6.3.2.43	CH_IRQ_PROLOGUE	32
6.3.2.44	CH_IRQ_EPILOGUE	32
6.3.2.45	CH_IRQ_HANDLER	32
6.3.2.46	CH_FAST_IRQ_HANDLER	32
6.3.2.47	TIME_S2I	33
6.3.2.48	TIME_MS2I	33
6.3.2.49	TIME_US2I	34
6.3.2.50	TIME_I2S	34
6.3.2.51	TIME_I2MS	35
6.3.2.52	TIME_I2US	35
6.3.2.53	_THREADS_QUEUE_DATA	36
6.3.2.54	_THREADS_QUEUE_DECL	36
6.3.2.55	_SEMAPHORE_DATA	36
6.3.2.56	SEMAPHORE_DECL	36

6.3.2.57	chSysGetRealtimeCounterX	37
6.3.2.58	chSysDisable	37
6.3.2.59	chSysSuspend	37
6.3.2.60	chSysEnable	38
6.3.2.61	chSysLock	38
6.3.2.62	chSysUnlock	38
6.3.2.63	chSysLockFromISR	39
6.3.2.64	chSysUnlockFromISR	39
6.3.2.65	chSchIsRescRequiredI	39
6.3.2.66	chThdGetSelfX	40
6.3.2.67	chThdSleepSeconds	40
6.3.2.68	chThdSleepMilliseconds	40
6.3.2.69	chThdSleepMicroseconds	40
6.3.2.70	chThdSleepS	41
6.3.2.71	chThdSleepUntilS	41
6.3.2.72	chThdQueueObjectInit	41
6.3.2.73	chThdQueueIsEmptyI	42
6.3.2.74	chSemObjectInit	42
6.3.2.75	chSemWait	42
6.3.2.76	chSemWaitS	43
6.3.2.77	chSemFastWaitI	43
6.3.2.78	chSemFastSignalI	44
6.3.2.79	chSemGetCounterI	44
6.3.2.80	chVTGetSystemTimeX	44
6.3.2.81	chVTTimeElapsedSinceX	44
6.3.2.82	chTimeAddX	45
6.3.2.83	chTimeDiffX	45
6.3.2.84	chTimeIsInRangeX	45
6.3.2.85	chDbgCheck	46
6.3.2.86	chDbgAssert	46
6.3.3	Typedef Documentation	47
6.3.3.1	sys_time_t	47
6.3.3.2	sys_interval_t	47
6.3.3.3	time_conv_t	47
6.3.3.4	thread_t	47
6.3.3.5	threads_queue_t	48
6.3.3.6	semaphore_t	48
6.3.3.7	tfunc_t	48
6.3.3.8	thread_config_t	48
6.3.3.9	thread_reference_t	48

6.3.3.10	<code>nil_system_t</code>	48
6.3.4	Function Documentation	48
6.3.4.1	<code>_dbg_check_disable(void)</code>	48
6.3.4.2	<code>_dbg_check_suspend(void)</code>	49
6.3.4.3	<code>_dbg_check_enable(void)</code>	49
6.3.4.4	<code>_dbg_check_lock(void)</code>	49
6.3.4.5	<code>_dbg_check_unlock(void)</code>	50
6.3.4.6	<code>_dbg_check_lock_from_isr(void)</code>	50
6.3.4.7	<code>_dbg_check_unlock_from_isr(void)</code>	51
6.3.4.8	<code>_dbg_check_enter_isr(void)</code>	51
6.3.4.9	<code>_dbg_check_leave_isr(void)</code>	51
6.3.4.10	<code>chDbgCheckClassI(void)</code>	52
6.3.4.11	<code>chDbgCheckClassS(void)</code>	52
6.3.4.12	<code>chSysInit(void)</code>	53
6.3.4.13	<code>chSysHalt(const char *reason)</code>	53
6.3.4.14	<code>chSysTimerHandlerI(void)</code>	54
6.3.4.15	<code>chSysUnconditionalLock(void)</code>	54
6.3.4.16	<code>chSysUnconditionalUnlock(void)</code>	54
6.3.4.17	<code>chSysGetStatusAndLockX(void)</code>	54
6.3.4.18	<code>chSysRestoreStatusX(syssts_t sts)</code>	55
6.3.4.19	<code>chSysIsCounterWithinX(rtcnt_t cnt, rtcnt_t start, rtcnt_t end)</code>	55
6.3.4.20	<code>chSysPolledDelayX(rtcnt_t cycles)</code>	56
6.3.4.21	<code>chSchReadyI(thread_t *tp, msg_t msg)</code>	56
6.3.4.22	<code>chSchIsPreemptionRequired(void)</code>	57
6.3.4.23	<code>chSchDoReschedule(void)</code>	57
6.3.4.24	<code>chSchRescheduleS(void)</code>	57
6.3.4.25	<code>chSchGoSleepTimeoutS(tstate_t newstate, sysinterval_t timeout)</code>	58
6.3.4.26	<code>chThdSuspendTimeoutS(thread_reference_t *trp, sysinterval_t timeout)</code>	59
6.3.4.27	<code>chThdResumeI(thread_reference_t *trp, msg_t msg)</code>	59
6.3.4.28	<code>chThdSleep(sysinterval_t timeout)</code>	60
6.3.4.29	<code>chThdSleepUntil(sysptime_t abstime)</code>	60
6.3.4.30	<code>chThdEnqueueTimeoutS(threads_queue_t *tqp, sysinterval_t timeout)</code>	60
6.3.4.31	<code>chThdDoDequeueNextI(threads_queue_t *tqp, msg_t msg)</code>	61
6.3.4.32	<code>chThdDequeueNextI(threads_queue_t *tqp, msg_t msg)</code>	62
6.3.4.33	<code>chThdDequeueAllI(threads_queue_t *tqp, msg_t msg)</code>	62
6.3.4.34	<code>chSemWaitTimeout(semaphore_t *sp, sysinterval_t timeout)</code>	62
6.3.4.35	<code>chSemWaitTimeoutS(semaphore_t *sp, sysinterval_t timeout)</code>	63
6.3.4.36	<code>chSemSignal(semaphore_t *sp)</code>	64
6.3.4.37	<code>chSemSignalI(semaphore_t *sp)</code>	64
6.3.4.38	<code>chSemReset(semaphore_t *sp, cnt_t n)</code>	65

6.3.4.39	chSemResetI(semaphore_t *sp, cnt_t n)	66
6.3.4.40	chEvtSignal(thread_t *tp, eventmask_t mask)	66
6.3.4.41	chEvtSignalI(thread_t *tp, eventmask_t mask)	67
6.3.4.42	chEvtWaitAnyTimeout(eventmask_t mask, sysinterval_t timeout)	68
6.3.5	Variable Documentation	68
6.3.5.1	nil	68
6.4	Objects_factory	69
6.4.1	Detailed Description	69
6.4.2	Macro Definition Documentation	71
6.4.2.1	CH_CFG_FACTORY_MAX_NAMES_LENGTH	71
6.4.2.2	CH_CFG_FACTORY_OBJECTS_REGISTRY	71
6.4.2.3	CH_CFG_FACTORY_GENERIC_BUFFERS	72
6.4.2.4	CH_CFG_FACTORY_SEMAPHORES	72
6.4.2.5	CH_CFG_FACTORY_SEMAPHORES	72
6.4.2.6	CH_CFG_FACTORY_MAILBOXES	72
6.4.2.7	CH_CFG_FACTORY_MAILBOXES	72
6.4.2.8	CH_CFG_FACTORY_OBJ_FIFOS	72
6.4.2.9	CH_CFG_FACTORY_OBJ_FIFOS	72
6.4.3	Typedef Documentation	72
6.4.3.1	dyn_element_t	72
6.4.3.2	dyn_list_t	72
6.4.3.3	registered_object_t	72
6.4.3.4	dyn_buffer_t	72
6.4.3.5	dyn_semaphore_t	72
6.4.3.6	dyn_mailbox_t	73
6.4.3.7	dyn_objects_fifo_t	73
6.4.3.8	objects_factory_t	73
6.4.4	Function Documentation	73
6.4.4.1	_factory_init(void)	73
6.4.4.2	chFactoryRegisterObject(const char *name, void *objp)	73
6.4.4.3	chFactoryFindObject(const char *name)	74
6.4.4.4	chFactoryFindObjectByPointer(void *objp)	74
6.4.4.5	chFactoryReleaseObject(registered_object_t *rop)	75
6.4.4.6	chFactoryCreateBuffer(const char *name, size_t size)	75
6.4.4.7	chFactoryFindBuffer(const char *name)	76
6.4.4.8	chFactoryReleaseBuffer(dyn_buffer_t *dbp)	76
6.4.4.9	chFactoryCreateSemaphore(const char *name, cnt_t n)	76
6.4.4.10	chFactoryFindSemaphore(const char *name)	77
6.4.4.11	chFactoryReleaseSemaphore(dyn_semaphore_t *dsp)	77
6.4.4.12	chFactoryCreateMailbox(const char *name, size_t n)	78

6.4.4.13	chFactoryFindMailbox(const char *name)	78
6.4.4.14	chFactoryReleaseMailbox(dyn_mailbox_t *dmp)	79
6.4.4.15	chFactoryCreateObjectsFIFO(const char *name, size_t objsize, size_t objn, unsigned objalign)	79
6.4.4.16	chFactoryFindObjectsFIFO(const char *name)	80
6.4.4.17	chFactoryReleaseObjectsFIFO(dyn_objects_fifo_t *dofp)	81
6.4.4.18	chFactoryDuplicateReference(dyn_element_t *dep)	81
6.4.4.19	chFactoryGetObject(registered_object_t *rop)	81
6.4.4.20	chFactoryGetBufferSize(dyn_buffer_t *dbp)	82
6.4.4.21	chFactoryGetBuffer(dyn_buffer_t *dbp)	82
6.4.4.22	chFactoryGetSemaphore(dyn_semaphore_t *dsp)	82
6.4.4.23	chFactoryGetMailbox(dyn_mailbox_t *dmp)	83
6.4.4.24	chFactoryGetObjectsFIFO(dyn_objects_fifo_t *dofp)	83
6.4.5	Variable Documentation	83
6.4.5.1	ch_factory	83
6.5	Heaps	84
6.5.1	Detailed Description	84
6.5.2	Macro Definition Documentation	85
6.5.2.1	CH_HEAP_ALIGNMENT	85
6.5.2.2	CH_HEAP_AREA	85
6.5.3	Typedef Documentation	85
6.5.3.1	memory_heap_t	85
6.5.3.2	heap_header_t	85
6.5.4	Function Documentation	85
6.5.4.1	_heap_init(void)	85
6.5.4.2	chHeapObjectInit(memory_heap_t *heapp, void *buf, size_t size)	86
6.5.4.3	chHeapAllocAligned(memory_heap_t *heapp, size_t size, unsigned align)	86
6.5.4.4	chHeapFree(void *p)	87
6.5.4.5	chHeapStatus(memory_heap_t *heapp, size_t *totalp, size_t *largestp)	87
6.5.4.6	chHeapAlloc(memory_heap_t *heapp, size_t size)	87
6.5.4.7	chHeapGetSize(const void *p)	88
6.5.5	Variable Documentation	88
6.5.5.1	default_heap	88
6.6	Mailboxes	89
6.6.1	Detailed Description	89
6.6.2	Macro Definition Documentation	90
6.6.2.1	_MAILBOX_DATA	90
6.6.2.2	MAILBOX_DECL	91
6.6.3	Function Documentation	91
6.6.3.1	chMBOBJECTInit(mailbox_t *mbp, msg_t *buf, size_t n)	91

6.6.3.2	chMBReset(mailbox_t *mbp)	91
6.6.3.3	chMBResetl(mailbox_t *mbp)	92
6.6.3.4	chMBPostTimeout(mailbox_t *mbp, msg_t msg, sysinterval_t timeout)	92
6.6.3.5	chMBPostTimeoutS(mailbox_t *mbp, msg_t msg, sysinterval_t timeout)	93
6.6.3.6	chMBPostl(mailbox_t *mbp, msg_t msg)	94
6.6.3.7	chMBPostAheadTimeout(mailbox_t *mbp, msg_t msg, sysinterval_t timeout)	95
6.6.3.8	chMBPostAheadTimeoutS(mailbox_t *mbp, msg_t msg, sysinterval_t timeout)	96
6.6.3.9	chMBPostAheadl(mailbox_t *mbp, msg_t msg)	97
6.6.3.10	chMBFetchTimeout(mailbox_t *mbp, msg_t *msgp, sysinterval_t timeout)	98
6.6.3.11	chMBFetchTimeoutS(mailbox_t *mbp, msg_t *msgp, sysinterval_t timeout)	99
6.6.3.12	chMBFetchl(mailbox_t *mbp, msg_t *msgp)	100
6.6.3.13	chMBGetSizel(const mailbox_t *mbp)	101
6.6.3.14	chMBGetUsedCountl(const mailbox_t *mbp)	101
6.6.3.15	chMBGetFreeCountl(const mailbox_t *mbp)	101
6.6.3.16	chMBPeekl(const mailbox_t *mbp)	102
6.6.3.17	chMBResumeX(mailbox_t *mbp)	103
6.7	Memcore	104
6.7.1	Detailed Description	104
6.7.2	Macro Definition Documentation	105
6.7.2.1	CH_CFG_MEMCORE_SIZE	105
6.7.3	Typedef Documentation	105
6.7.3.1	memgetfunc_t	105
6.7.3.2	memgetfunc2_t	105
6.7.4	Function Documentation	105
6.7.4.1	_core_init(void)	105
6.7.4.2	chCoreAllocAlignedWithOffsetl(size_t size, unsigned align, size_t offset)	106
6.7.4.3	chCoreAllocAlignedWithOffset(size_t size, unsigned align, size_t offset)	106
6.7.4.4	chCoreGetStatusX(void)	107
6.7.4.5	chCoreAllocAlignedl(size_t size, unsigned align)	107
6.7.4.6	chCoreAllocAligned(size_t size, unsigned align)	108
6.7.4.7	chCoreAlloc(size_t size)	108
6.7.4.8	chCoreAlloc(size_t size)	109
6.7.5	Variable Documentation	110
6.7.5.1	ch_memcore	110
6.8	Pools	111
6.8.1	Detailed Description	111
6.8.2	Macro Definition Documentation	112
6.8.2.1	_MEMORYPOOL_DATA	112
6.8.2.2	MEMORYPOOL_DECL	113
6.8.2.3	_GUARDEDMEMORYPOOL_DATA	113

6.8.2.4	GUARDEDMEMORYPOOL_DECL	113
6.8.3	Function Documentation	113
6.8.3.1	chPoolObjectInitAligned(memory_pool_t *mp, size_t size, unsigned align, memgetfunc_t provider)	113
6.8.3.2	chPoolLoadArray(memory_pool_t *mp, void *p, size_t n)	114
6.8.3.3	chPoolAlloc(memory_pool_t *mp)	114
6.8.3.4	chPoolAlloc(memory_pool_t *mp)	115
6.8.3.5	chPoolFree(memory_pool_t *mp, void *objp)	116
6.8.3.6	chPoolFree(memory_pool_t *mp, void *objp)	116
6.8.3.7	chGuardedPoolObjectInitAligned(guarded_memory_pool_t *gmp, size_t size, unsigned align)	117
6.8.3.8	chGuardedPoolLoadArray(guarded_memory_pool_t *gmp, void *p, size_t n)	118
6.8.3.9	chGuardedPoolAllocTimeoutS(guarded_memory_pool_t *gmp, sysinterval_t timeout)	118
6.8.3.10	chGuardedPoolAllocTimeout(guarded_memory_pool_t *gmp, sysinterval_t timeout)	119
6.8.3.11	chGuardedPoolFree(guarded_memory_pool_t *gmp, void *objp)	120
6.8.3.12	chGuardedPoolFree(guarded_memory_pool_t *gmp, void *objp)	120
6.8.3.13	chPoolObjectInit(memory_pool_t *mp, size_t size, memgetfunc_t provider)	121
6.8.3.14	chPoolAdd(memory_pool_t *mp, void *objp)	122
6.8.3.15	chPoolAdd(memory_pool_t *mp, void *objp)	122
6.8.3.16	chGuardedPoolObjectInit(guarded_memory_pool_t *gmp, size_t size)	123
6.8.3.17	chGuardedPoolAdd(guarded_memory_pool_t *gmp, void *objp)	124
6.8.3.18	chGuardedPoolAdd(guarded_memory_pool_t *gmp, void *objp)	124
6.8.3.19	chGuardedPoolAlloc(guarded_memory_pool_t *gmp)	125
6.9	Binary semaphores	127
6.9.1	Detailed Description	127
6.9.2	Macro Definition Documentation	128
6.9.2.1	_BSEMAPHORE_DATA	128
6.9.2.2	BSEMAPHORE_DECL	128
6.9.3	Typedef Documentation	128
6.9.3.1	binary_semaphore_t	128
6.9.4	Function Documentation	128
6.9.4.1	chBSemObjectInit(binary_semaphore_t *bsp, bool taken)	128
6.9.4.2	chBSemWait(binary_semaphore_t *bsp)	129
6.9.4.3	chBSemWaitS(binary_semaphore_t *bsp)	129
6.9.4.4	chBSemWaitTimeoutS(binary_semaphore_t *bsp, sysinterval_t timeout)	130
6.9.4.5	chBSemWaitTimeout(binary_semaphore_t *bsp, sysinterval_t timeout)	131
6.9.4.6	chBSemReset(binary_semaphore_t *bsp, bool taken)	131
6.9.4.7	chBSemReset(binary_semaphore_t *bsp, bool taken)	132
6.9.4.8	chBSemSignal(binary_semaphore_t *bsp)	133

6.9.4.9	chBSemSignal(binary_semaphore_t *bsp)	133
6.9.4.10	chBSemGetStatel(const binary_semaphore_t *bsp)	134
6.10	Objects_fifo	135
6.10.1	Detailed Description	135
6.10.2	Typedef Documentation	135
6.10.2.1	objects_fifo_t	135
6.10.3	Function Documentation	136
6.10.3.1	chFifoObjectInit(objects_fifo_t *ofp, size_t objsize, size_t objn, unsigned objalign, void *objbuf, msg_t *msgbuf)	136
6.10.3.2	chFifoTakeObjectI(objects_fifo_t *ofp)	136
6.10.3.3	chFifoTakeObjectTimeoutS(objects_fifo_t *ofp, sysinterval_t timeout)	137
6.10.3.4	chFifoTakeObjectTimeout(objects_fifo_t *ofp, sysinterval_t timeout)	138
6.10.3.5	chFifoReturnObjectI(objects_fifo_t *ofp, void *objp)	138
6.10.3.6	chFifoReturnObject(objects_fifo_t *ofp, void *objp)	139
6.10.3.7	chFifoSendObjectI(objects_fifo_t *ofp, void *objp)	139
6.10.3.8	chFifoSendObjectS(objects_fifo_t *ofp, void *objp)	140
6.10.3.9	chFifoSendObject(objects_fifo_t *ofp, void *objp)	141
6.10.3.10	chFifoReceiveObjectI(objects_fifo_t *ofp, void **objpp)	141
6.10.3.11	chFifoReceiveObjectTimeoutS(objects_fifo_t *ofp, void **objpp, sysinterval_t timeout)	142
6.10.3.12	chFifoReceiveObjectTimeout(objects_fifo_t *ofp, void **objpp, sysinterval_t timeout)	143
7	Data Structure Documentation	145
7.1	ch_binary_semaphore Struct Reference	145
7.1.1	Detailed Description	146
7.2	ch_dyn_element Struct Reference	146
7.2.1	Detailed Description	147
7.2.2	Field Documentation	147
7.2.2.1	next	147
7.2.2.2	refs	147
7.3	ch_dyn_list Struct Reference	147
7.3.1	Detailed Description	147
7.4	ch_dyn_mailbox Struct Reference	148
7.4.1	Detailed Description	148
7.4.2	Field Documentation	149
7.4.2.1	element	149
7.4.2.2	mbx	149
7.4.2.3	msgbuf	149
7.5	ch_dyn_object Struct Reference	149
7.5.1	Detailed Description	150

7.5.2	Field Documentation	150
7.5.2.1	element	150
7.5.2.2	buffer	150
7.6	ch_dyn_objects_fifo Struct Reference	150
7.6.1	Detailed Description	151
7.6.2	Field Documentation	151
7.6.2.1	element	151
7.6.2.2	fifo	152
7.6.2.3	msgbuf	152
7.7	ch_dyn_semaphore Struct Reference	152
7.7.1	Detailed Description	152
7.7.2	Field Documentation	153
7.7.2.1	element	153
7.7.2.2	sem	153
7.8	ch_objects_factory Struct Reference	153
7.8.1	Detailed Description	154
7.8.2	Field Documentation	154
7.8.2.1	mtx	154
7.8.2.2	obj_list	154
7.8.2.3	obj_pool	154
7.8.2.4	buf_list	154
7.8.2.5	sem_list	154
7.8.2.6	sem_pool	154
7.8.2.7	mbx_list	154
7.8.2.8	fifo_list	155
7.9	ch_objects_fifo Struct Reference	155
7.9.1	Detailed Description	156
7.9.2	Field Documentation	156
7.9.2.1	free	156
7.9.2.2	mbx	156
7.10	ch_registered_static_object Struct Reference	156
7.10.1	Detailed Description	157
7.10.2	Field Documentation	157
7.10.2.1	element	157
7.10.2.2	objp	157
7.11	guarded_memory_pool_t Struct Reference	157
7.11.1	Detailed Description	158
7.11.2	Field Documentation	158
7.11.2.1	sem	158
7.11.2.2	pool	158

7.12 heap_header Union Reference	158
7.12.1 Detailed Description	159
7.12.2 Field Documentation	159
7.12.2.1 next	159
7.12.2.2 pages	159
7.12.2.3 heap	159
7.12.2.4 size	159
7.13 mailbox_t Struct Reference	159
7.13.1 Detailed Description	160
7.13.2 Field Documentation	160
7.13.2.1 buffer	160
7.13.2.2 top	160
7.13.2.3 wrptr	160
7.13.2.4 rdptr	160
7.13.2.5 cnt	160
7.13.2.6 reset	160
7.13.2.7 qw	161
7.13.2.8 qr	161
7.14 memcore_t Struct Reference	161
7.14.1 Detailed Description	161
7.14.2 Field Documentation	161
7.14.2.1 nextmem	161
7.14.2.2 endmem	161
7.15 memory_heap Struct Reference	162
7.15.1 Detailed Description	162
7.15.2 Field Documentation	162
7.15.2.1 provider	162
7.15.2.2 header	162
7.15.2.3 mtx	163
7.16 memory_pool_t Struct Reference	163
7.16.1 Detailed Description	163
7.16.2 Field Documentation	163
7.16.2.1 next	163
7.16.2.2 object_size	164
7.16.2.3 align	164
7.16.2.4 provider	164
7.17 nil_system Struct Reference	164
7.17.1 Detailed Description	166
7.17.2 Field Documentation	166
7.17.2.1 current	166

7.17.2.2	next	166
7.17.2.3	sysptime	166
7.17.2.4	lasttime	166
7.17.2.5	nexttime	166
7.17.2.6	isr_cnt	166
7.17.2.7	lock_cnt	166
7.17.2.8	dbg_panic_msg	167
7.17.2.9	threads	167
7.18	nil_thread Struct Reference	167
7.18.1	Detailed Description	168
7.18.2	Field Documentation	168
7.18.2.1	ctx	168
7.18.2.2	state	168
7.18.2.3	msg	168
7.18.2.4	p	168
7.18.2.5	trp	168
7.18.2.6	tqp	168
7.18.2.7	semp	169
7.18.2.8	ewmask	169
7.18.2.9	timeout	169
7.18.2.10	epmask	169
7.18.2.11	wabase	169
7.19	nil_thread_cfg Struct Reference	169
7.19.1	Detailed Description	170
7.19.2	Field Documentation	170
7.19.2.1	wbase	170
7.19.2.2	wend	170
7.19.2.3	namep	170
7.19.2.4	funcp	170
7.19.2.5	arg	170
7.20	nil_threads_queue Struct Reference	170
7.20.1	Detailed Description	171
7.20.2	Field Documentation	171
7.20.2.1	cnt	171
7.21	pool_header Struct Reference	172
7.21.1	Detailed Description	172
7.21.2	Field Documentation	172
7.21.2.1	next	172

8.1	ch.c File Reference	173
8.1.1	Detailed Description	175
8.2	ch.h File Reference	175
8.2.1	Detailed Description	181
8.3	chbsem.h File Reference	181
8.3.1	Detailed Description	182
8.4	chconf.h File Reference	182
8.4.1	Detailed Description	184
8.5	chfactory.c File Reference	184
8.5.1	Detailed Description	185
8.6	chfactory.h File Reference	185
8.6.1	Detailed Description	187
8.7	chfifo.h File Reference	187
8.7.1	Detailed Description	188
8.8	chheap.c File Reference	188
8.8.1	Detailed Description	189
8.9	chheap.h File Reference	189
8.9.1	Detailed Description	190
8.10	chmboxes.c File Reference	190
8.10.1	Detailed Description	191
8.11	chmboxes.h File Reference	191
8.11.1	Detailed Description	192
8.12	chmemcore.c File Reference	192
8.12.1	Detailed Description	192
8.13	chmemcore.h File Reference	192
8.13.1	Detailed Description	193
8.14	chmempools.c File Reference	193
8.14.1	Detailed Description	194
8.15	chmempools.h File Reference	194
8.15.1	Detailed Description	196
Index		197

Chapter 1

ChibiOS/NIL

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/NIL portable Kernel.

1.3 Related Documents

- ChibiOS/NIL General Architecture

Chapter 2

Module Index

2.1 Modules

Here is a list of all modules:

NIL Kernel	11
Configuration	12
API	19
Objects_factory	69
Heaps	84
Mailboxes	89
Memcore	104
Pools	111
Binary_semaphores	127
Objects_fifo	135

Chapter 3

Hierarchical Index

3.1 Class Hierarchy

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

ch_dyn_element	146
ch_dyn_list	147
ch_dyn_mailbox	148
ch_dyn_object	149
ch_dyn_objects_fifo	150
ch_dyn_semaphore	152
ch_objects_factory	153
ch_objects_fifo	155
ch_registered_static_object	156
guarded_memory_pool_t	157
heap_header	158
mailbox_t	159
memcore_t	161
memory_heap	162
memory_pool_t	163
nil_system	164
nil_thread	167
nil_thread_cfg	169
nil_threads_queue	170
ch_binary_semaphore	145
pool_header	172

Chapter 4

Data Structure Index

4.1 Data Structures

Here are the data structures with brief descriptions:

ch_binary_semaphore	Binary semaphore type	145
ch_dyn_element	Type of a dynamic object list element	146
ch_dyn_list	Type of a dynamic object list	147
ch_dyn_mailbox	Type of a dynamic buffer object	148
ch_dyn_object	Type of a dynamic buffer object	149
ch_dyn_objects_fifo	Type of a dynamic buffer object	150
ch_dyn_semaphore	Type of a dynamic semaphore	152
ch_objects_factory	Type of the factory main object	153
ch_objects_fifo	Type of an objects FIFO	155
ch_registered_static_object	Type of a registered object	156
guarded_memory_pool_t	Guarded memory pool descriptor	157
heap_header	Memory heap block header	158
mailbox_t	Structure representing a mailbox object	159
memcore_t	Type of memory core object	161
memory_heap	Structure describing a memory heap	162
memory_pool_t	Memory pool descriptor	163
nil_system	System data structure	164
nil_thread	Structure representing a thread	167
nil_thread_cfg	Structure representing a thread static configuration	169

nil_threads_queue	
Structure representing a queue of threads	170
pool_header	
Memory pool free object header	172

Chapter 5

File Index

5.1 File List

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

ch.c	Nil RTOS main source file	173
ch.h	Nil RTOS main header file	175
chbsem.h	Binary semaphores structures and macros	181
chconf.h	Configuration file template	182
chfactory.c	ChibiOS objects factory and registry code	184
chfactory.h	ChibiOS objects factory structures and macros	185
chfifo.h	Objects FIFO structures and macros	187
chheap.c	Heaps code	188
chheap.h	Heaps macros and structures	189
chmbboxes.c	Mailboxes code	190
chmbboxes.h	Mailboxes macros and structures	191
chmemcore.c	Core memory manager code	192
chmemcore.h	Core memory manager macros and structures	192
chmempools.c	Memory Pools code	193
chmempools.h	Memory Pools macros and structures	194

Chapter 6

Module Documentation

6.1 NIL Kernel

6.1.1 Detailed Description

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

Modules

- [Configuration](#)
- [API](#)

6.2 Configuration

6.2.1 Detailed Description

Kernel related settings and hooks.

Kernel parameters and options

- `#define CH_CFG_NUM_THREADS 3`
Number of user threads in the application.

System timer settings

- `#define CH_CFG_ST_RESOLUTION 32`
System time counter resolution.
- `#define CH_CFG_ST_FREQUENCY 1000`
System tick frequency.
- `#define CH_CFG_ST_TIMEDELTA 0`
Time delta constant for the tick-less mode.

Subsystem options

- `#define CH_CFG_USE_SEMAPHORES TRUE`
Semaphores APIs.
- `#define CH_CFG_USE_MUTEXES FALSE`
Mutexes APIs.
- `#define CH_CFG_USE_EVENTS TRUE`
Events Flags APIs.
- `#define CH_CFG_USE_MAILBOXES TRUE`
Mailboxes APIs.
- `#define CH_CFG_USE_MEMCORE TRUE`
Core Memory Manager APIs.
- `#define CH_CFG_USE_HEAP TRUE`
Heap Allocator APIs.
- `#define CH_CFG_USE_MEMPOOLS TRUE`
Memory Pools Allocator APIs.
- `#define CH_CFG_USE_OBJ_FIFOS TRUE`
Objects FIFOs APIs.
- `#define CH_CFG_MEMCORE_SIZE 0`
Managed RAM size.

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.*

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
System assertions.
- #define CH_DBG_ENABLE_STACK_CHECK TRUE
Stack check.

Kernel hooks

- #define CH_CFG_SYSTEM_INIT_HOOK()
System initialization hook.
- #define CH_CFG_THREAD_EXT_FIELDS /* Add threads custom fields here.*/
Threads descriptor structure extension.
- #define CH_CFG_THREAD_EXT_INIT_HOOK(tr)
Threads initialization hook.
- #define CH_CFG_IDLE_ENTER_HOOK()
Idle thread enter hook.
- #define CH_CFG_IDLE_LEAVE_HOOK()
Idle thread leave hook.
- #define CH_CFG_SYSTEM_HALT_HOOK(reason)
System halt hook.

6.2.2 Macro Definition Documentation

6.2.2.1 #define CH_CFG_NUM_THREADS 3

Number of user threads in the application.

Note

This number is not inclusive of the idle thread which is Implicitly handled.

6.2.2.2 #define CH_CFG_ST_RESOLUTION 32

System time counter resolution.

Note

Allowed values are 16 or 32 bits.

6.2.2.3 `#define CH_CFG_ST_FREQUENCY 1000`

System tick frequency.

Note

This value together with the `CH_CFG_ST_RESOLUTION` option defines the maximum amount of time allowed for timeouts.

6.2.2.4 `#define CH_CFG_ST_TIMEDELTA 0`

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.

6.2.2.5 `#define CH_CFG_USE_SEMAPHORES TRUE`

Semaphores APIs.

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

Note

The default is `TRUE`.

6.2.2.6 `#define CH_CFG_USE_MUTEXES FALSE`

Mutexes APIs.

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

Note

Feature not currently implemented.
The default is `FALSE`.

6.2.2.7 `#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`.

6.2.2.8 `#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`.

6.2.2.9 #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`.

6.2.2.10 #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`.

6.2.2.11 #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`.

6.2.2.12 #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`.

6.2.2.13 #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`.

6.2.2.14 #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`.

6.2.2.15 #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.

6.2.2.16 #define CH_CFG_FACTORY_OBJECTS_REGISTRY TRUE

Enables the registry of generic objects.

6.2.2.17 #define CH_CFG_FACTORY_GENERIC_BUFFERS TRUE

Enables factory for generic buffers.

6.2.2.18 #define CH_CFG_FACTORY_SEMAPHORES TRUE

Enables factory for semaphores.

6.2.2.19 #define CH_CFG_FACTORY_MAILBOXES TRUE

Enables factory for mailboxes.

6.2.2.20 #define CH_CFG_FACTORY_OBJ_FIFOS TRUE

Enables factory for objects FIFOs.

6.2.2.21 #define CH_DBG_STATISTICS FALSE

Debug option, kernel statistics.

Note

Feature not currently implemented.

The default is `FALSE`.

6.2.2.22 #define CH_DBG_SYSTEM_STATE_CHECK TRUE

Debug option, system state check.

Note

The default is `FALSE`.

6.2.2.23 #define CH_DBG_ENABLE_CHECKS TRUE

Debug option, parameters checks.

Note

The default is `FALSE`.

6.2.2.24 #define CH_DBG_ENABLE_ASSERTS TRUE

System assertions.

Note

The default is `FALSE`.

6.2.2.25 #define CH_DBG_ENABLE_STACK_CHECK TRUE

Stack check.

Note

The default is `FALSE`.

6.2.2.26 #define CH_CFG_SYSTEM_INIT_HOOK()**Value:**

```
{
    \
}
```

System initialization hook.

6.2.2.27 #define CH_CFG_THREAD_EXT_FIELDS /* Add threads custom fields here.*/

Threads descriptor structure extension.

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

6.2.2.28 #define CH_CFG_THREAD_EXT_INIT_HOOK(tr)**Value:**

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

Threads initialization hook.

6.2.2.29 #define CH_CFG_IDLE_ENTER_HOOK()**Value:**

```
{                                     \
}
```

Idle thread enter hook.

Note

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

6.2.2.30 #define CH_CFG_IDLE_LEAVE_HOOK()**Value:**

```
{                                     \
}
```

Idle thread leave hook.

Note

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

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

```
{                                     \
}
```

System halt hook.

6.3 API

6.3.1 Detailed Description

Macros

- `#define _CHIBIOS_NIL_`
ChibiOS/NIL identification macro.
- `#define CH_KERNEL_STABLE 1`
Stable release flag.
- `#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 THD_IDLE_BASE (&__main_thread_stack_base__)`
- `#define __CH_STRINGIFY(a) #a`
Utility to make the parameter a quoted string.

ChibiOS/NIL version identification

- `#define CH_KERNEL_VERSION "3.0.0"`
Kernel version string.
- `#define CH_KERNEL_MAJOR 3`
Kernel version major number.
- `#define CH_KERNEL_MINOR 0`
Kernel version minor number.
- `#define CH_KERNEL_PATCH 0`
Kernel version patch number.

Wakeup messages

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

Special time constants

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

Thread state related macros

- `#define NIL_STATE_READY (tstate_t)0`
Thread ready or executing.
- `#define NIL_STATE_SLEEPING (tstate_t)1`
Thread sleeping.
- `#define NIL_STATE_SUSP (tstate_t)2`
Thread suspended.
- `#define NIL_STATE_WTQUEUE (tstate_t)3`
On queue or semaph.
- `#define NIL_STATE_WTOREVT (tstate_t)4`
Waiting for events.
- `#define NIL_THD_IS_READY(tr) ((tr)->state == NIL_STATE_READY)`
- `#define NIL_THD_IS_SLEEPING(tr) ((tr)->state == NIL_STATE_SLEEPING)`
- `#define NIL_THD_IS_SUSP(tr) ((tr)->state == NIL_STATE_SUSP)`
- `#define NIL_THD_IS_WTQUEUE(tr) ((tr)->state == NIL_STATE_WTQUEUE)`
- `#define NIL_THD_IS_WTOREVT(tr) ((tr)->state == NIL_STATE_WTOREVT)`

Events related macros

- `#define ALL_EVENTS ((eventmask_t)-1)`
All events allowed mask.
- `#define EVENT_MASK(eid) ((eventmask_t)(1 << (eid)))`
Returns an event mask from an event identifier.

Threads tables definition macros

- `#define THD_TABLE_BEGIN const thread_config_t nil_thd_configs[CH_CFG_NUM_THREADS + 1] = {`
Start of user threads table.
- `#define THD_TABLE_ENTRY(wap, name, funcp, arg)`
Entry of user threads table.
- `#define THD_TABLE_END`
End of user threads table.

Memory alignment support macros

- #define `MEM_ALIGN_MASK(a)` `((size_t)(a) - 1U)`
Alignment mask constant.
- #define `MEM_ALIGN_PREV(p, a)` `((size_t)(p) & ~MEM_ALIGN_MASK(a))`
Aligns to the previous aligned memory address.
- #define `MEM_ALIGN_NEXT(p, a)`
Aligns to the new 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.

Working Areas

- #define `THD_WORKING_AREA_SIZE(n)`
Calculates the total Working Area size.
- #define `THD_WORKING_AREA(s, n)` `PORT_WORKING_AREA(s, n)`
Static working area allocation.

Threads abstraction macros

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

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

- #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.

Threads queues

- #define `_THREADS_QUEUE_DATA`(name) {(cnt_t)0}
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.

Semaphores macros

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

Macro Functions

- #define `chSysGetRealtimeCounterX`() (rtcnt_t)port_rt_get_counter_value()
Returns the current value of the system real time counter.
- #define `chSysDisable`()
Raises the system interrupt priority mask to the maximum level.
- #define `chSysSuspend`()
Raises the system interrupt priority mask to system level.
- #define `chSysEnable`()
Lowers the system interrupt priority mask to user level.
- #define `chSysLock`()
Enters the kernel lock state.
- #define `chSysUnlock`()
Leaves the kernel lock state.
- #define `chSysLockFromISR`()
Enters the kernel lock state from within an interrupt handler.
- #define `chSysUnlockFromISR`()
Leaves the kernel lock state from within an interrupt handler.
- #define `chSchIsRescRequiredI`() ((bool)(nil.current != nil.next))
Evaluates if a reschedule is required.
- #define `chThdGetSelfX`() nil.current
Returns a pointer to the current `thread_t`.
- #define `chThdSleepSeconds`(secs) `chThdSleep`(`TIME_S2I`(secs))
Delays the invoking thread for the specified number of seconds.
- #define `chThdSleepMilliseconds`(msecs) `chThdSleep`(`TIME_MS2I`(msecs))
Delays the invoking thread for the specified number of milliseconds.
- #define `chThdSleepMicroseconds`(usecs) `chThdSleep`(`TIME_US2I`(usecs))
Delays the invoking thread for the specified number of microseconds.

- #define `chThdSleepS`(timeout) (void) `chSchGoSleepTimeoutS`(NIL_STATE_SLEEPING, timeout)
Suspends the invoking thread for the specified time.
- #define `chThdSleepUntilS`(abstime)
Suspends the invoking thread until the system time arrives to the specified value.
- #define `chThdQueueObjectInit`(tqp) ((tqp)->cnt = (cnt_t)0)
Initializes a threads queue object.
- #define `chThdQueueIsEmptyI`(tqp) ((bool)(tqp->cnt >= (cnt_t)0))
Evaluates to `true` if the specified queue is empty.
- #define `chSemObjectInit`(sp, n) ((sp)->cnt = n)
Initializes a semaphore with the specified counter value.
- #define `chSemWait`(sp) `chSemWaitTimeout`(sp, TIME_INFINITE)
Performs a wait operation on a semaphore.
- #define `chSemWaitS`(sp) `chSemWaitTimeoutS`(sp, TIME_INFINITE)
Performs a wait operation on a semaphore.
- #define `chSemFastWaitI`(sp) ((sp)->cnt--)
Decreases the semaphore counter.
- #define `chSemFastSignalI`(sp) ((sp)->cnt++)
Increases the semaphore counter.
- #define `chSemGetCounterI`(sp) ((sp)->cnt)
Returns the semaphore counter current value.
- #define `chVTGetSystemTimeX`() (nil.systime)
Current system time.
- #define `chVTimeElapsedSinceX`(start) `chTimeDiffX`((start), `chVTGetSystemTimeX`())
Returns the elapsed time since the specified start time.
- #define `chTimeAddX`(systime, interval) ((`systime_t`)(systime) + (`systime_t`)(interval))
Adds an interval to a system time returning a system time.
- #define `chTimeDiffX`(start, end) ((`sysinterval_t`)((`systime_t`)(end) - (`systime_t`)(start)))
Subtracts two system times returning an interval.
- #define `chTimeIsInRangeX`(time, start, end)
Checks if the specified time is within the specified time range.
- #define `chDbgCheck`(c)
Function parameters check.
- #define `chDbgAssert`(c, r)
Condition assertion.

Typedefs

- typedef uint32_t `systime_t`
Type of system time.
- typedef uint32_t `sysinterval_t`
Type of time interval.
- typedef uint64_t `time_conv_t`
Type of time conversion variable.
- typedef struct `nil_thread` `thread_t`
Type of a structure representing a thread.
- typedef struct `nil_threads_queue` `threads_queue_t`
Type of a queue of threads.
- typedef `threads_queue_t` `semaphore_t`
Type of a structure representing a semaphore.
- typedef void(* `tfunc_t`) (void *p)

Thread function.

- typedef struct [nil_thread_cfg](#) [thread_config_t](#)
Type of a structure representing a thread static configuration.
- typedef [thread_t](#) * [thread_reference_t](#)
Type of a thread reference.
- typedef struct [nil_system](#) [nil_system_t](#)
Type of a structure representing the system.

Data Structures

- struct [nil_threads_queue](#)
Structure representing a queue of threads.
- struct [nil_thread_cfg](#)
Structure representing a thread static configuration.
- struct [nil_thread](#)
Structure representing a thread.
- struct [nil_system](#)
System 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.
- void [chSysInit](#) (void)
Initializes the kernel.
- void [chSysHalt](#) (const char *reason)
Halts the system.
- void [chSysTimerHandlerI](#) (void)
Time management handler.
- void [chSysUnconditionalLock](#) (void)

- Unconditionally enters the kernel lock state.*
- void `chSysUnconditionalUnlock` (void)
- Unconditionally leaves the kernel lock state.*
- `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.*
- `thread_t` * `chSchReadyI` (`thread_t` *tp, `msg_t` msg)
- Makes the specified thread ready for execution.*
- bool `chSchIsPreemptionRequired` (void)
- Evaluates if preemption is required.*
- void `chSchDoReschedule` (void)
- Switches to the first thread on the runnable queue.*
- void `chSchRescheduleS` (void)
- Reschedules if needed.*
- `msg_t` `chSchGoSleepTimeoutS` (`tstate_t` newstate, `sysinterval_t` timeout)
- Puts the current thread to sleep into the specified state with timeout specification.*
- `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 `chThdSleep` (`sysinterval_t` timeout)
- Suspends the invoking thread for the specified time.*
- void `chThdSleepUntil` (`systime_t` abstime)
- Suspends the invoking thread until the system time arrives to the specified value.*
- `msg_t` `chThdEnqueueTimeoutS` (`threads_queue_t` *tqp, `sysinterval_t` timeout)
- Enqueues the caller thread on a threads queue object.*
- void `chThdDoDequeueNextI` (`threads_queue_t` *tqp, `msg_t` msg)
- Dequeues and wakes up one thread from the 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.*
- `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 `chSemSignalI` (`semaphore_t` *sp)
- Performs a signal operation on a semaphore.*
- void `chSemReset` (`semaphore_t` *sp, `cnt_t` n)
- Performs a reset operation on the semaphore.*
- void `chSemResetI` (`semaphore_t` *sp, `cnt_t` n)
- Performs a reset operation on the semaphore.*
- void `chEvtSignal` (`thread_t` *tp, `eventmask_t` mask)
- Adds a set of event flags directly to the specified thread_t.*

- void `chEvtSignal`(`thread_t` *tp, `eventmask_t` mask)
Adds a set of event flags directly to the specified `thread_t`.
- `eventmask_t` `chEvtWaitAnyTimeout`(`eventmask_t` mask, `sysinterval_t` timeout)
Waits for any of the specified events.

Variables

- `nil_system_t` nil
System data structures.

6.3.2 Macro Definition Documentation

6.3.2.1 `#define _CHIBIOS_NIL_`

ChibiOS/NIL identification macro.

6.3.2.2 `#define CH_KERNEL_STABLE 1`

Stable release flag.

6.3.2.3 `#define CH_KERNEL_VERSION "3.0.0"`

Kernel version string.

6.3.2.4 `#define CH_KERNEL_MAJOR 3`

Kernel version major number.

6.3.2.5 `#define CH_KERNEL_MINOR 0`

Kernel version minor number.

6.3.2.6 `#define CH_KERNEL_PATCH 0`

Kernel version patch number.

6.3.2.7 `#define MSG_OK (msg_t)0`

OK wakeup message.

6.3.2.8 `#define MSG_TIMEOUT (msg_t)-1`

Wake-up caused by a timeout condition.

6.3.2.9 `#define MSG_RESET (msg_t)-2`

Wake-up caused by a reset condition.

6.3.2.10 #define TIME_IMMEDIATE ((sysinterval_t)-1)

Zero time specification for some functions with a timeout specification.

Note

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

6.3.2.11 #define TIME_INFINITE ((sysinterval_t)0)

Infinite time specification for all functions with a timeout specification.

6.3.2.12 #define TIME_MAX_INTERVAL ((sysinterval_t)-2)

Maximum interval constant usable as timeout.

6.3.2.13 #define TIME_MAX_SYSTIME ((systime_t)-1)

Maximum system of system time before it wraps.

6.3.2.14 #define NIL_STATE_READY (tstate_t)0

Thread ready or executing.

6.3.2.15 #define NIL_STATE_SLEEPING (tstate_t)1

Thread sleeping.

6.3.2.16 #define NIL_STATE_SUSP (tstate_t)2

Thread suspended.

6.3.2.17 #define NIL_STATE_WTQUEUE (tstate_t)3

On queue or semaph.

6.3.2.18 #define NIL_STATE_WTOREVT (tstate_t)4

Waiting for events.

6.3.2.19 #define ALL_EVENTS ((eventmask_t)-1)

All events allowed mask.

6.3.2.20 #define EVENT_MASK(eid) ((eventmask_t)(1 << (eid)))

Returns an event mask from an event identifier.

6.3.2.21 `#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`.

6.3.2.22 `#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.

6.3.2.23 `#define CH_CFG_FACTORY_OBJECTS_REGISTRY TRUE`

Enables the registry of generic objects.

6.3.2.24 `#define CH_CFG_FACTORY_GENERIC_BUFFERS TRUE`

Enables factory for generic buffers.

6.3.2.25 `#define CH_CFG_FACTORY_SEMAPHORES TRUE`

Enables factory for semaphores.

6.3.2.26 `#define CH_CFG_FACTORY_MAILBOXES TRUE`

Enables factory for mailboxes.

6.3.2.27 `#define CH_CFG_FACTORY_OBJ_FIFOS TRUE`

Enables factory for objects FIFOs.

6.3.2.28 `#define THD_IDLE_BASE (&__main_thread_stack_base__)`

Boundaries of the idle thread boundaries, only required if stack checking is enabled.

6.3.2.29 `#define __CH_STRINGIFY(a) #a`

Utility to make the parameter a quoted string.

6.3.2.30 `#define THD_TABLE_BEGIN const thread_config_t nil_thd_configs[CH_CFG_NUM_THREADS + 1] = {`

Start of user threads table.

6.3.2.31 #define THD_TABLE_ENTRY(wap, name, funcp, arg)**Value:**

```
{wap, ((stkalign_t *) (wap)) + (sizeof (wap) / sizeof (stkalign_t)), \
    name, funcp, arg},
```

Entry of user threads table.

6.3.2.32 #define THD_TABLE_END**Value:**

```
{THD_IDLE_BASE, THD_IDLE_END, "idle", NULL, NULL} \
};
```

End of user threads table.

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

Alignment mask constant.

Parameters

in	<i>a</i>	alignment, must be a power of two
----	----------	-----------------------------------

6.3.2.34 #define MEM_ALIGN_PREV(p, a)((size_t)(p) & ~MEM_ALIGN_MASK(a))

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

6.3.2.35 #define MEM_ALIGN_NEXT(p, a)**Value:**

```
MEM_ALIGN_PREV((size_t)(p) + \
    MEM_ALIGN_MASK(a), (a))
```

Aligns to the new aligned memory address.

Parameters

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

6.3.2.36 `#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

6.3.2.37 `#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
----	----------	---

6.3.2.38 `#define THD_WORKING_AREA_SIZE(n)`

Value:

`MEM_ALIGN_NEXT(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.

6.3.2.39 `#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.

6.3.2.40 `#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.

6.3.2.41 `#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

<i>in</i>	<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.

6.3.2.42 `#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

<i>in</i>	<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.

6.3.2.43 #define CH_IRQ_PROLOGUE()

Value:

```
PORT_IRQ_PROLOGUE();
_dbg_check_enter_isr();
```

IRQ handler enter code.

Note

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

Function Class:

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

6.3.2.44 #define CH_IRQ_EPILOGUE()

Value:

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

IRQ handler exit code.

Note

Usually IRQ handlers function are also declared naked.

Function Class:

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

6.3.2.45 #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.

6.3.2.46 #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.

6.3.2.47 `#define TIME_S2I(secs)((sysinterval_t)((time_conv_t)(secs) * (time_conv_t)CH_CFG_ST_FREQUENCY ← CY))`

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	<i>secs</i>	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.

6.3.2.48 `#define TIME_MS2I(msec)`

Value:

```
((sysinterval_t) (((time_conv_t) (msec) *
                    (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	<i>msec</i>	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.

6.3.2.49 #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.

6.3.2.50 #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.

6.3.2.51 #define TIME_I2MS(*interval*)**Value:**

```
(time_msecs_t) ((( (time_conv_t) (interval) * (time_conv_t) 1000) +
                  (time_conv_t) CH_CFG_ST_FREQUENCY - (
time_conv_t) 1) / \
                  (time_conv_t) 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	<i>interval</i>	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.

6.3.2.52 #define TIME_I2US(*interval*)**Value:**

```
(time_msecs_t) ((( (time_conv_t) (interval) * (time_conv_t) 1000000) +
                  (time_conv_t) CH_CFG_ST_FREQUENCY - (
time_conv_t) 1) / \
                  (time_conv_t) 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.

6.3.2.53 #define _THREADS_QUEUE_DATA(*name*) {{cnt_t}0}

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
----	-------------	--

6.3.2.54 #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
----	-------------	--

6.3.2.55 #define _SEMAPHORE_DATA(*name*, *n*) {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

6.3.2.56 #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
----	-------------	------------------------------------

Parameters

in	<i>n</i>	the counter initial value, this value must be non-negative
----	----------	--

6.3.2.57 #define chSysGetRealtimeCounterX() (rtcnt_t)port_rt_get_counter_value()

Returns the current value of the system real time counter.

Note

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

Returns

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

Function Class:

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

6.3.2.58 #define chSysDisable()

Value:

```
{
    port_disable();
    _dbg_check_disable();
}
```

Raises the system interrupt priority mask to the maximum level.

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

Note

Do not invoke this API from within a kernel lock.

Function Class:

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

6.3.2.59 #define chSysSuspend()

Value:

```
{
    port_suspend();
    _dbg_check_suspend();
}
```

Raises the system interrupt priority mask to system level.

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

Note

Do not invoke this API from within a kernel lock.

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

Function Class:

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

6.3.2.60 #define chSysEnable()**Value:**

```
{
    _dbg_check_enable();
    port_enable();
}
```

Lowers the system interrupt priority mask to user level.

All the interrupt sources are enabled.

Note

Do not invoke this API from within a kernel lock.

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

Function Class:

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

6.3.2.61 #define chSysLock()**Value:**

```
{
    port_lock();
    _dbg_check_lock();
}
```

Enters the kernel lock state.

Function Class:

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

6.3.2.62 #define chSysUnlock()**Value:**

```
{
    _dbg_check_unlock();
    port_unlock();
}
```

Leaves the kernel lock state.

Function Class:

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

6.3.2.63 #define chSysLockFromISR()

Value:

```

{
    port_lock_from_isr();
    _dbg_check_lock_from_isr();
}

```

Enters the kernel lock state from within an interrupt handler.

Note

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

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

This API must be invoked exclusively from interrupt handlers.

Function Class:

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

6.3.2.64 #define chSysUnlockFromISR()

Value:

```

{
    _dbg_check_unlock_from_isr();
    port_unlock_from_isr();
}

```

Leaves the kernel lock state from within an interrupt handler.

Note

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

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

This API must be invoked exclusively from interrupt handlers.

Function Class:

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

6.3.2.65 #define chSchIsRescRequired() ((bool)(nil.current != nil.next))

Evaluates if a reschedule is required.

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:

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

handlers.

6.3.2.66 `#define chThdGetSelfX() nil.current`

Returns a pointer to the current `thread_t`.

Function Class:

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

6.3.2.67 `#define chThdSleepSeconds(secs) chThdSleep(TIME_S2I(secs))`

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 clock.
The maximum specified value is implementation dependent.

Parameters

in	<i>secs</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.

6.3.2.68 `#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 clock.
The maximum specified value is implementation dependent.

Parameters

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

Function Class:

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

6.3.2.69 `#define chThdSleepMicroseconds(usecs) chThdSleep(TIME_US2I(usecs))`

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 clock.
The maximum specified value is implementation dependent.

Parameters

in	<i>usecs</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.

6.3.2.70 `#define chThdSleepS(timeout) (void) chSchGoSleepTimeoutS(NIL_STATE_SLEEPING, timeout)`

Suspends the invoking thread for the specified time.

Parameters

in	<i>timeout</i>	the delay in system ticks
----	----------------	---------------------------

Function Class:

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

6.3.2.71 `#define chThdSleepUntilS(abstime)`

Value:

```
(void) chSchGoSleepTimeoutS(NIL_STATE_SLEEPING,
    \
    chTimeDiffX(chVTGetSystemTimeX(), (abstime)))
```

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

Parameters

in	<i>abstime</i>	absolute system time
----	----------------	----------------------

Function Class:

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

6.3.2.72 `#define chThdQueueObjectInit(tqp) ((tqp)->cnt = (cnt_t)0)`

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.

6.3.2.73 `#define chThdQueueIsEmpty(tqp) ((bool)(tqp->cnt >= (cnt_t)0))`

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

Parameters

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

Returns

The queue status.

Return values

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

Function Class:

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

6.3.2.74 `#define chSemObjectInit(sp, n) ((sp)->cnt = 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.

6.3.2.75 `#define chSemWait(sp) chSemWaitTimeout(sp, TIME_INFINITE)`

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

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

Function Class:

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

6.3.2.76 #define chSemWaitS(sp) chSemWaitTimeoutS(sp, TIME_INFINITE)

Performs a wait operation on a semaphore.

Parameters

in	sp	pointer to a <code>semaphore_t</code> structure
----	----	---

Returns

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

Return values

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

Function Class:

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

6.3.2.77 #define chSemFastWaitl(sp) ((sp)->cnt--)

Decreases the semaphore counter.

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

Parameters

in	sp	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.

6.3.2.78 `#define chSemFastSignal(sp) ((sp)->cnt++)`

Increases the semaphore counter.

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

Parameters

in	sp	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.

6.3.2.79 `#define chSemGetCounter(sp) ((sp)->cnt)`

Returns the semaphore counter current value.

Function Class:

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

6.3.2.80 `#define chVTGetSystemTimeX() (nil.systime)`

Current system time.

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

Note

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

This function 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.

6.3.2.81 `#define chVTTimeElapsedSinceX(start) chTimeDiffX((start), chVTGetSystemTimeX())`

Returns the elapsed time since the specified start time.

Parameters

in	start	start time
----	-------	------------

Returns

The elapsed time.

Function Class:

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

6.3.2.82 `#define chTimeAddX(sysTime, interval) ((sysTime_t)(sysTime) + (sysTime_t)(interval))`

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.

6.3.2.83 `#define chTimeDiffX(start, end) ((sysInterval_t)((sysTime_t)((sysTime_t)(end) - (sysTime_t)(start))))`

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.

6.3.2.84 `#define chTimeIsInRangeX(time, start, end)`

Value:

```
((bool)((sysTime_t)((sysTime_t)(time) - (sysTime_t)(start)) <
      (sysTime_t)((sysTime_t)(end) - (sysTime_t)(start)))) \
```

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

Note

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

Parameters

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

Return values

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

Function Class:

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

6.3.2.85 #define chDbgCheck(c)**Value:**

```
do {
    /*lint -save -e506 -e774 [2.1, 14.3] Can be a constant by design.*/
    if (CH_DBG_ENABLE_CHECKS != FALSE) {
        if (!(c)) {
            /*lint -restore*/
            chSysHalt(__func__);
        }
    }
} while (false)
```

Function parameters check.

If the condition check fails then the kernel panics and halts.

Note

The condition is tested only if the CH_DBG_ENABLE_CHECKS switch is specified in `chconf.h` else the macro does nothing.

Parameters

in	<i>c</i>	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.

6.3.2.86 #define chDbgAssert(c, r)**Value:**

```
do {
    /*lint -save -e506 -e774 [2.1, 14.3] Can be a constant by design.*/
    if (CH_DBG_ENABLE_ASSERTS != FALSE) {
        if (!(c)) {
            /*lint -restore*/
            chSysHalt(__func__);
        }
    }
}
```

```

}
} while (false)

```

Condition assertion.

If the condition check fails then the kernel panics with a message and halts.

Note

The condition is tested only if the `CH_DBG_ENABLE_ASSERTS` switch is specified in `chconf.h` else the macro does nothing.

The remark string is not currently used except for putting a comment in the code about the assertion.

Parameters

in	<i>c</i>	the condition to be verified to be true
in	<i>r</i>	a remark string

Function Class:

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

6.3.3 Typedef Documentation

6.3.3.1 typedef uint32_t systime_t

Type of system time.

Note

It is selectable in configuration between 16 or 32 bits.

6.3.3.2 typedef uint32_t sysinterval_t

Type of time interval.

Note

It is selectable in configuration between 16 or 32 bits.

6.3.3.3 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.

6.3.3.4 typedef struct nil_thread thread_t

Type of a structure representing a thread.

Note

It is required as an early definition.

6.3.3.5 `typedef struct nil_threads_queue threads_queue_t`

Type of a queue of threads.

6.3.3.6 `typedef threads_queue_t semaphore_t`

Type of a structure representing a semaphore.

Note

Semaphores are implemented on thread queues, the object is the same, the behavior is slightly different.

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

Thread function.

6.3.3.8 `typedef struct nil_thread_cfg thread_config_t`

Type of a structure representing a thread static configuration.

6.3.3.9 `typedef thread_t* thread_reference_t`

Type of a thread reference.

6.3.3.10 `typedef struct nil_system nil_system_t`

Type of a structure representing the system.

6.3.4 Function Documentation

6.3.4.1 `void _dbg_check_disable (void)`

Guard code for `chSysDisable()`.

Function Class:

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

Here is the call graph for this function:



6.3.4.2 void _dbg_check_suspend (void)

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

Function Class:

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

Here is the call graph for this function:



6.3.4.3 void _dbg_check_enable (void)

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

Function Class:

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

Here is the call graph for this function:



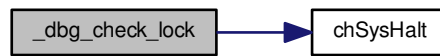
6.3.4.4 void _dbg_check_lock (void)

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

Function Class:

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

Here is the call graph for this function:



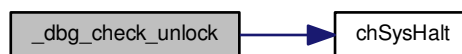
6.3.4.5 void _dbg_check_unlock (void)

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

Function Class:

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

Here is the call graph for this function:



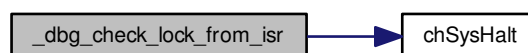
6.3.4.6 void _dbg_check_lock_from_isr (void)

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

Function Class:

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

Here is the call graph for this function:



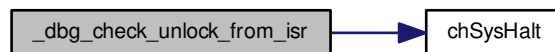
6.3.4.7 void _dbg_check_unlock_from_isr (void)

Guard code for `chSysUnlockFromIsr()`.

Function Class:

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

Here is the call graph for this function:



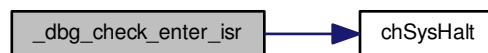
6.3.4.8 void _dbg_check_enter_isr (void)

Guard code for `CH_IRQ_PROLOGUE()`.

Function Class:

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

Here is the call graph for this function:



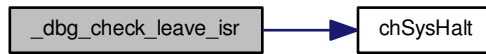
6.3.4.9 void _dbg_check_leave_isr (void)

Guard code for `CH_IRQ_EPILOGUE()`.

Function Class:

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

Here is the call graph for this function:



6.3.4.10 void chDbgCheckClassI (void)

I-class functions context check.

Verifies that the system is in an appropriate state for invoking an I-class API function. A panic is generated if the state is not compatible.

Function Class:

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

Here is the call graph for this function:



6.3.4.11 void chDbgCheckClassS (void)

S-class functions context check.

Verifies that the system is in an appropriate state for invoking an S-class API function. A panic is generated if the state is not compatible.

Function Class:

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

Here is the call graph for this function:



6.3.4.12 void chSysInit (void)

Initializes the kernel.

Initializes the kernel structures, the current instructions flow becomes the idle thread upon return. The idle thread must not invoke any kernel primitive able to change state to not runnable.

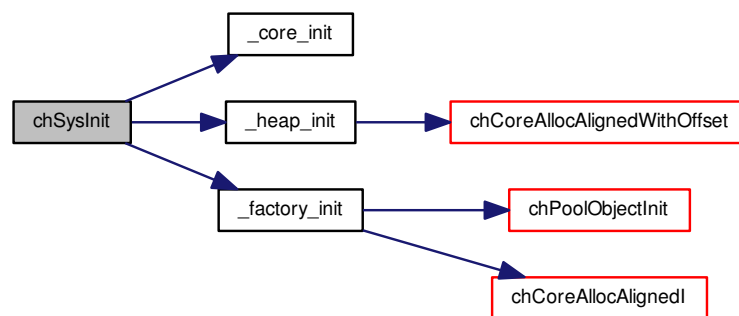
Note

This function assumes that the `nil` global variable has been zeroed by the runtime environment. If this is not the case then make sure to clear it before calling this function.

Function Class:

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

Here is the call graph for this function:



6.3.4.13 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.

6.3.4.14 void chSysTimerHandlerI (void)

Time management handler.

Note

This handler has to be invoked by a periodic ISR in order to reschedule the waiting threads.

Function 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:



6.3.4.15 void chSysUnconditionalLock (void)

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.

6.3.4.16 void chSysUnconditionalUnlock (void)

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.

6.3.4.17 syssts_t chSysGetStatusAndLockX (void)

Returns the execution status and enters a critical zone.

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

Postcondition

The system is in a critical zone.

Returns

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

Function Class:

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

6.3.4.18 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.

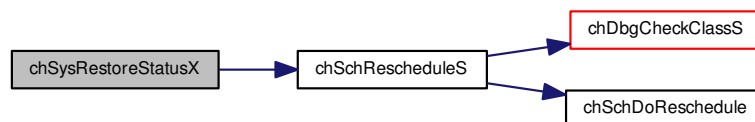
Parameters

in	sts	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:

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

Realtime window test.

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

Note

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

Parameters

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

Return values

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

Function Class:

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

6.3.4.20 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:

**6.3.4.21 thread_t * chSchReadyI (thread_t * tp, msg_t msg)**

Makes the specified thread ready for execution.

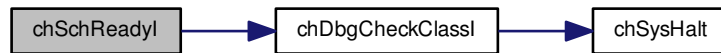
Parameters

in	<i>tp</i>	pointer to the <code>thread_t</code> object
in	<i>msg</i>	the wakeup message

Returns

The same reference passed as parameter.

Here is the call graph for this function:

**6.3.4.22 bool chSchIsPreemptionRequired (void)**

Evaluates if preemption is required.

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

Note

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

Return values

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

Function Class:

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

6.3.4.23 void chSchDoReschedule (void)

Switches to the first thread on the runnable queue.

Note

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

Function Class:

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

6.3.4.24 void chSchRescheduleS (void)

Reschedules if needed.

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:



6.3.4.25 `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 system time then it is forcibly awakened with a `NIL_MSG_TMO` low level message.

Parameters

in	<i>newstate</i>	the new thread state or a semaphore pointer
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_INFINITE</code> no timeout.

Returns

The wakeup message.

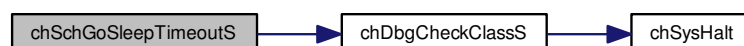
Return values

<code>NIL_MSG_TMO</code>	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:



6.3.4.26 `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 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.

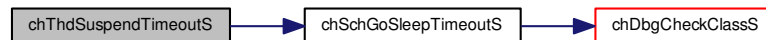
Returns

The 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:



6.3.4.27 `void chThdResume1 (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.

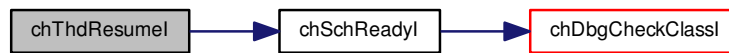
Parameters

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:



6.3.4.28 void chThdSleep (sysinterval_t timeout)

Suspends the invoking thread for the specified time.

Parameters

in	<i>timeout</i>	the delay in system ticks
----	----------------	---------------------------

Function Class:

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

6.3.4.29 void chThdSleepUntil (systime_t abstime)

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

Parameters

in	<i>abstime</i>	absolute system time
----	----------------	----------------------

Function Class:

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

6.3.4.30 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 <i>MSG_TIMEOUT</i> as if a timeout occurred.

Returns

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

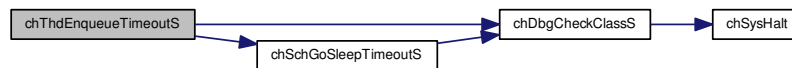
Return values

<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:



6.3.4.31 void chThdDoDequeueNextI (threads_queue_t * *tqp*, msg_t *msg*)

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.

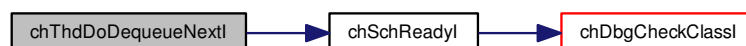
Parameters

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:



6.3.4.32 void chThdDequeueNextI (threads_queue_t * *tqp*, msg_t *msg*)

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

Parameters

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

Function Class:

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

Here is the call graph for this function:



6.3.4.33 void chThdDequeueAllI (threads_queue_t * *tqp*, msg_t *msg*)

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

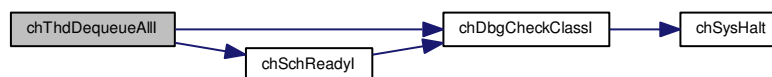
Parameters

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:



6.3.4.34 msg_t chSemWaitTimeout (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.

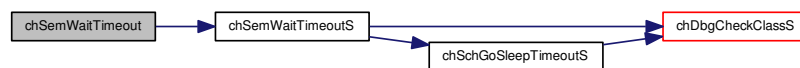
Return values

<code>NIL_MSG_OK</code>	if the thread has not stopped on the semaphore or the semaphore has been signaled.
<code>NIL_MSG_RST</code>	if the semaphore has been reset using <code>chSemReset()</code> .
<code>NIL_MSG_TMO</code>	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:

6.3.4.35 `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.

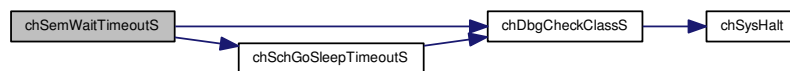
Return values

<i>NIL_MSG_OK</i>	if the thread has not stopped on the semaphore or the semaphore has been signaled.
<i>NIL_MSG_RST</i>	if the semaphore has been reset using chSemReset () .
<i>NIL_MSG_TMO</i>	if the semaphore has not been signaled or reset within the specified timeout.

Function Class:

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

Here is the call graph for this function:



6.3.4.36 void chSemSignal (semaphore_t * sp)

Performs a signal operation on a semaphore.

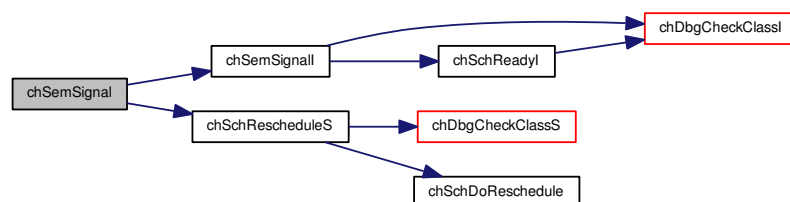
Parameters

in	sp	pointer to a semaphore_t structure
----	----	------------------------------------

Function Class:

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

Here is the call graph for this function:



6.3.4.37 void chSemSignal1 (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.

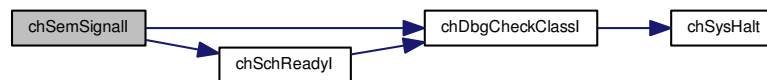
Parameters

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:

**6.3.4.38 void chSemReset (semaphore_t * *sp*, cnt_t *n*)**

Performs a reset operation on the semaphore.

Postcondition

After invoking this function all the threads waiting on the semaphore, if any, are released and the semaphore counter is set to the specified, non negative, value.

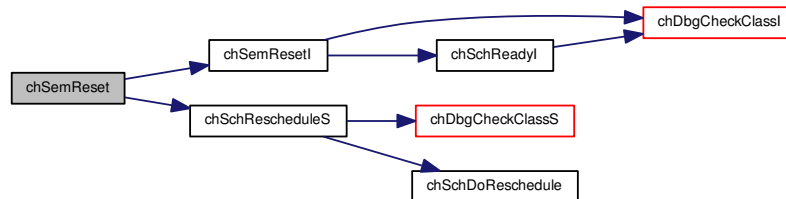
Parameters

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:



6.3.4.39 void chSemResetI (semaphore_t * sp, cnt_t n)

Performs a reset operation on the semaphore.

Postcondition

After invoking this function all the threads waiting on the semaphore, if any, are released and the semaphore counter is set to the specified, non negative, value.

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

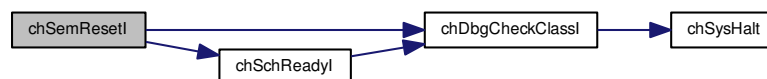
Parameters

in	sp	pointer to a semaphore_t structure
in	n	the new value of the semaphore counter. The value must be non-negative.

Function Class:

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

Here is the call graph for this function:



6.3.4.40 void chEvtSignal (thread_t * tp, eventmask_t mask)

Adds a set of event flags directly to the specified thread_t.

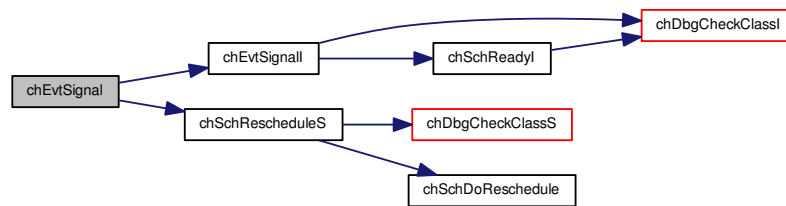
Parameters

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

Function Class:

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

Here is the call graph for this function:

**6.3.4.41 void chEvtSignall (thread_t * tp, eventmask_t mask)**

Adds a set of event flags directly to the specified `thread_t`.

Postcondition

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

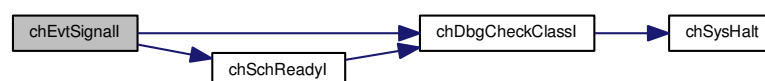
Parameters

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

Function Class:

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

Here is the call graph for this function:



6.3.4.42 eventmask_t chEvtWaitAnyTimeout (eventmask_t mask, sysinterval_t timeout)

Waits for any of the specified events.

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

Parameters

in	<i>mask</i>	mask of the event flags 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.

Returns

The mask of the served and cleared events.

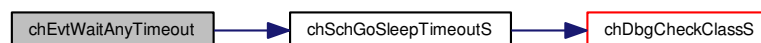
Return values

0	if the operation has timed out.
---	---------------------------------

Function Class:

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

Here is the call graph for this function:



6.3.5 Variable Documentation

6.3.5.1 nil_system_t nil

System data structures.

6.4 Objects_factory

6.4.1 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 FALSE`
Enables factory for objects FIFOs.

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_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_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 * chFactoryFindObjectsFIFO` (const char *name)
Retrieves a dynamic "objects FIFO" object.
- void `chFactoryReleaseObjectsFIFO` (dyn_objects_fifo_t *dofp)
Releases a dynamic "objects FIFO" 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.

Variables

- `objects_factory_t ch_factory`
Factory object static instance.

6.4.2 Macro Definition Documentation

6.4.2.1 #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.

6.4.2.2 #define CH_CFG_FACTORY_OBJECTS_REGISTRY TRUE

Enables the registry of generic objects.

6.4.2.3 `#define CH_CFG_FACTORY_GENERIC_BUFFERS TRUE`

Enables factory for generic buffers.

6.4.2.4 `#define CH_CFG_FACTORY_SEMAPHORES TRUE`

Enables factory for semaphores.

6.4.2.5 `#define CH_CFG_FACTORY_SEMAPHORES FALSE`

Enables factory for semaphores.

6.4.2.6 `#define CH_CFG_FACTORY_MAILBOXES TRUE`

Enables factory for mailboxes.

6.4.2.7 `#define CH_CFG_FACTORY_MAILBOXES FALSE`

Enables factory for mailboxes.

6.4.2.8 `#define CH_CFG_FACTORY_OBJ_FIFOS TRUE`

Enables factory for objects FIFOs.

6.4.2.9 `#define CH_CFG_FACTORY_OBJ_FIFOS FALSE`

Enables factory for objects FIFOs.

6.4.3 Typedef Documentation

6.4.3.1 `typedef struct ch_dyn_element dyn_element_t`

Type of a dynamic object list element.

6.4.3.2 `typedef struct ch_dyn_list dyn_list_t`

Type of a dynamic object list.

6.4.3.3 `typedef struct ch_registered_static_object registered_object_t`

Type of a registered object.

6.4.3.4 `typedef struct ch_dyn_object dyn_buffer_t`

Type of a dynamic buffer object.

6.4.3.5 `typedef struct ch_dyn_semaphore dyn_semaphore_t`

Type of a dynamic semaphore.

6.4.3.6 `typedef struct ch_dyn_mailbox dyn_mailbox_t`

Type of a dynamic buffer object.

6.4.3.7 `typedef struct ch_dyn_objects_fifo dyn_objects_fifo_t`

Type of a dynamic buffer object.

6.4.3.8 `typedef struct ch_objects_factory objects_factory_t`

Type of the factory main object.

6.4.4 Function Documentation

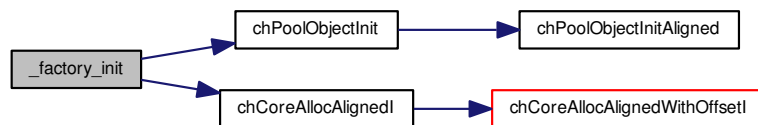
6.4.4.1 `void _factory_init (void)`

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:

6.4.4.2 `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

<i>NULL</i>	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.

6.4.4.3 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

in	<i>name</i>	name of the registered object
----	-------------	-------------------------------

Returns

The reference to the found registered object.

Return values

<i>NULL</i>	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.

6.4.4.4 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

<i>NULL</i>	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.

6.4.4.5 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.

6.4.4.6 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.

6.4.4.7 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	name	name of the dynamic buffer object
----	------	-----------------------------------

Returns

The reference to the found dynamic buffer object.

Return values

NULL	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.

6.4.4.8 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	dbp	dynamic buffer object reference
----	-----	---------------------------------

Function Class:

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

6.4.4.9 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

<i>NULL</i>	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.

6.4.4.10 `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

in	<i>name</i>	name of the dynamic semaphore object
----	-------------	--------------------------------------

Returns

The reference to the found dynamic semaphore object.

Return values

<i>NULL</i>	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.

6.4.4.11 `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.

6.4.4.12 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

<i>NULL</i>	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.

Here is the call graph for this function:

**6.4.4.13 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.

6.4.4.14 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.

6.4.4.15 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.

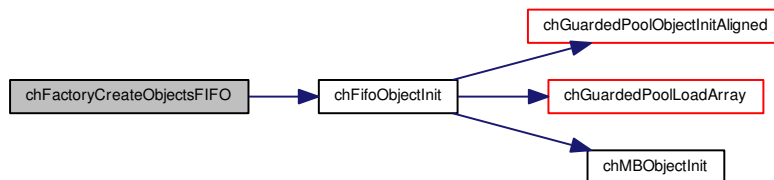
Return values

<i>NULL</i>	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.

Here is the call graph for this function:



6.4.4.16 `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

<i>NULL</i>	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.

6.4.4.17 void chFactoryReleaseObjectsFIFO (dyn_objects_fifo_t * *dofp*)

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>dofp</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.

6.4.4.18 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	<i>dep</i>	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.

6.4.4.19 static void* chFactoryGetObject (registered_object_t * *rop*) [inline], [static]

Returns the pointer to the inner registered object.

Parameters

in	<i>rop</i>	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.

6.4.4.20 `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
----	------------	---------------------------------

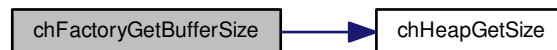
Returns

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:



6.4.4.21 `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.

6.4.4.22 `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.

6.4.4.23 `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.

6.4.4.24 `static objects_fifo_t* chFactoryGetObjectsFIFO (dyn_objects_fifo_t* dofp)` `[inline],[static]`

Returns the pointer to the inner objects FIFO.

Parameters

in	<i>dofp</i>	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.

6.4.5 Variable Documentation**6.4.5.1** `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.

6.5 Heaps

6.5.1 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.

6.5.2 Macro Definition Documentation

6.5.2.1 #define CH_HEAP_ALIGNMENT 8U

Minimum alignment used for heap.

Note

Cannot use the sizeof operator in this macro.

6.5.2.2 #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.

6.5.3 Typedef Documentation

6.5.3.1 typedef struct memory_heap memory_heap_t

Type of a memory heap.

6.5.3.2 typedef union heap_header heap_header_t

Type of a memory heap header.

6.5.4 Function Documentation

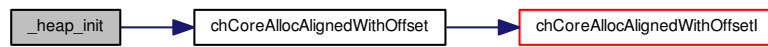
6.5.4.1 void _heap_init (void)

Initializes the default heap.

Function Class:

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

Here is the call graph for this function:



6.5.4.2 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 means 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.

6.5.4.3 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.

6.5.4.4 void chHeapFree (void * *p*)

Frees a previously allocated memory block.

Parameters

in	<i>p</i>	pointer to the memory block to be freed
----	----------	---

Function Class:

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

6.5.4.5 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 <code>NULL</code> in order to access the default heap.
in	<i>totalp</i>	pointer to a variable that will receive the total fragmented free space or <code>NULL</code>
in	<i>largestp</i>	pointer to a variable that will receive the largest free free block found space or <code>NULL</code>

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.

6.5.4.6 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

<i>NULL</i>	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:



6.5.4.7 `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.

6.5.5 Variable Documentation

6.5.5.1 `memory_heap_t default_heap [static]`

Default heap descriptor.

6.6 Mailboxes

6.6.1 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 `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 chMBPostI (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.*
- `static size_t chMBGetSizeI (const mailbox_t *mbp)`
- Returns the mailbox buffer size as number of messages.*
- `static size_t chMBGetUsedCountI (const mailbox_t *mbp)`
- Returns the number of used message slots into a mailbox.*
- `static size_t chMBGetFreeCountI (const mailbox_t *mbp)`
- Returns the number of free message slots into a mailbox.*
- `static msg_t chMBPeekI (const mailbox_t *mbp)`
- Returns the next message in the queue without removing it.*
- `static void chMBResumeX (mailbox_t *mbp)`
- Terminates the reset state.*

6.6.2 Macro Definition Documentation

6.6.2.1 #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 <code>msg_t</code>
in	<i>size</i>	number of <code>msg_t</code> elements in the buffer array

6.6.2.2 `#define MAILBOX_DECL(name, buffer, size) mailbox_t name = _MAILBOX_DATA(name, buffer, size)`

Static mailbox initializer.

Statically initialized mailboxes require no explicit initialization using `chMBOBJECTInit()`.

Parameters

in	<i>name</i>	the name of the mailbox variable
in	<i>buffer</i>	pointer to the mailbox buffer array of <code>msg_t</code>
in	<i>size</i>	number of <code>msg_t</code> elements in the buffer array

6.6.3 Function Documentation

6.6.3.1 `void chMBOBJECTInit(mailbox_t * mbp, msg_t * buf, size_t n)`

Initializes a `mailbox_t` object.

Parameters

out	<i>mbp</i>	the pointer to the <code>mailbox_t</code> structure to be initialized
in	<i>buf</i>	pointer to the messages buffer as an array of <code>msg_t</code>
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.

6.6.3.2 `void chMBReset(mailbox_t * mbp)`

Resets a `mailbox_t` object.

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

Postcondition

The mailbox is in reset state, all operations will fail and return `MSG_RESET` until the mailbox is enabled again using `chMBResumeX()`.

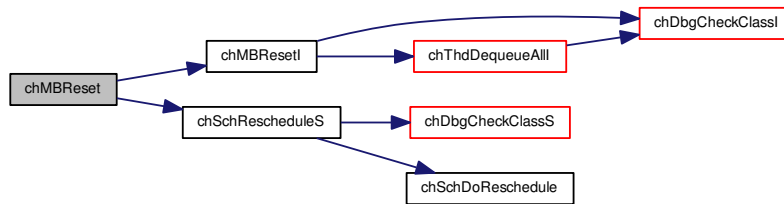
Parameters

in	<i>mbp</i>	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:



6.6.3.3 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()`.

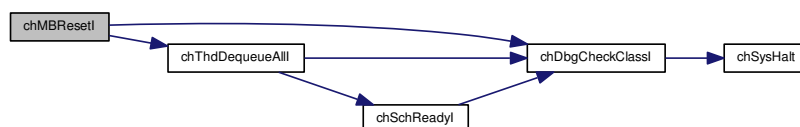
Parameters

in	<i>mbp</i>	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:



6.6.3.4 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
----	------------	---

Parameters

in	<i>msg</i>	the message to be posted on the mailbox
in	<i>timeout</i>	the number of ticks before the operation timeouts, the following special values are allowed: <ul style="list-style-type: none"> • <i>TIME_IMMEDIATE</i> immediate timeout. • <i>TIME_INFINITE</i> no timeout.

Returns

The operation status.

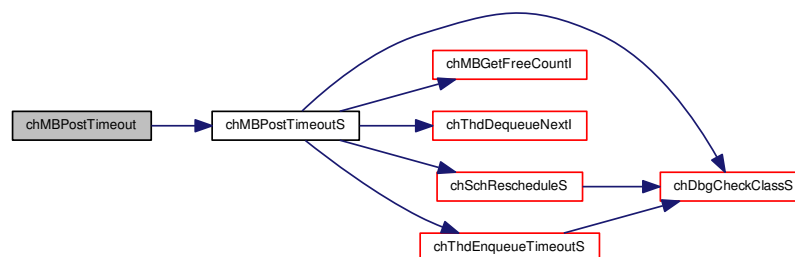
Return values

<i>MSG_OK</i>	if a message has been correctly posted.
<i>MSG_RESET</i>	if the mailbox has been reset.
<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:



6.6.3.5 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	<i>mbp</i>	the pointer to an initialized mailbox_t object
in	<i>msg</i>	the message to be posted on the mailbox
in	<i>timeout</i>	the number of ticks before the operation timeouts, the following special values are allowed: <ul style="list-style-type: none"> • <i>TIME_IMMEDIATE</i> immediate timeout. • <i>TIME_INFINITE</i> no timeout.

Returns

The operation status.

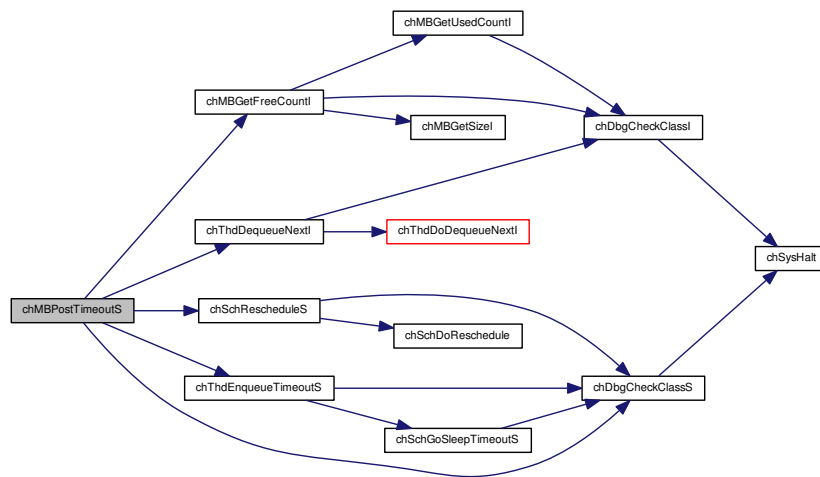
Return values

<i>MSG_OK</i>	if a message has been correctly posted.
<i>MSG_RESET</i>	if the mailbox has been reset.
<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:

**6.6.3.6 msg_t chMBPostI (mailbox_t * mbp, msg_t msg)**

Posts a message into a mailbox.

This variant is non-blocking, the function returns a timeout condition if the queue is full.

Parameters

in	<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.
---------------	---

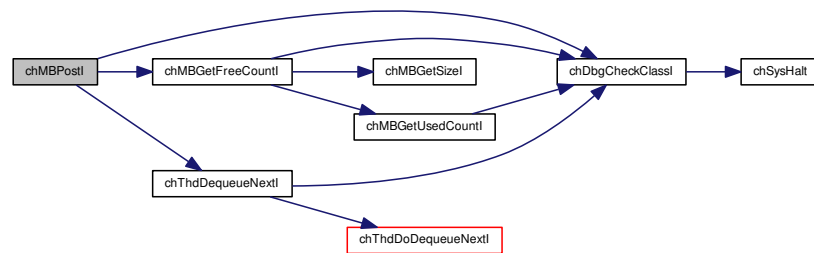
Return values

<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:



6.6.3.7 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 mailbox_t object
in	<i>msg</i>	the message to be posted on the mailbox
in	<i>timeout</i>	the number of ticks before the operation timeouts, the following special values are allowed: <ul style="list-style-type: none"> • <i>TIME_IMMEDIATE</i> immediate timeout. • <i>TIME_INFINITE</i> no timeout.

Returns

The operation status.

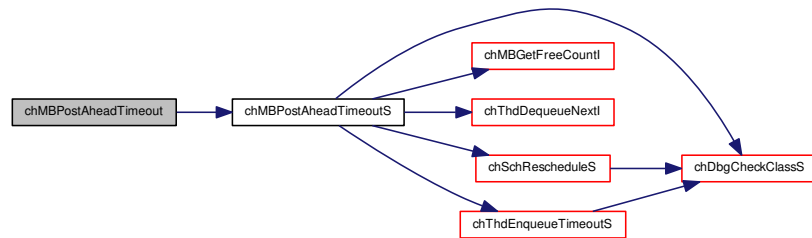
Return values

<i>MSG_OK</i>	if a message has been correctly posted.
<i>MSG_RESET</i>	if the mailbox has been reset.
<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:



6.6.3.8 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 mailbox_t object
in	<i>msg</i>	the message to be posted on the mailbox
in	<i>timeout</i>	the number of ticks before the operation timeouts, the following special values are allowed: <ul style="list-style-type: none"> • <i>TIME_IMMEDIATE</i> immediate timeout. • <i>TIME_INFINITE</i> no timeout.

Returns

The operation status.

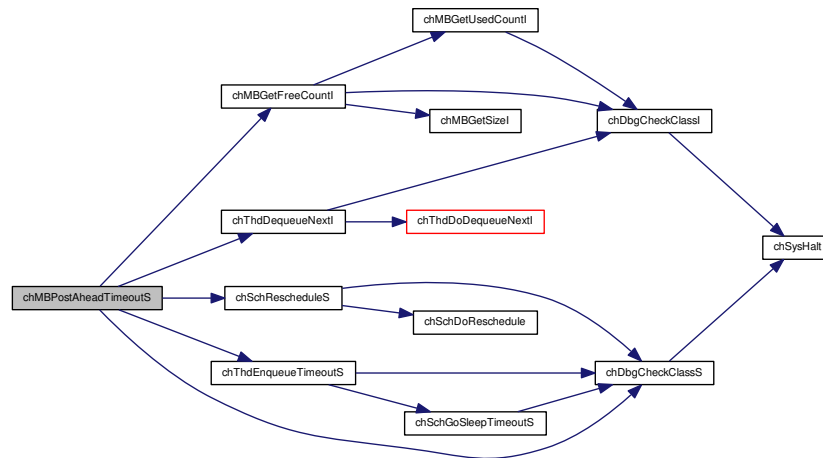
Return values

<i>MSG_OK</i>	if a message has been correctly posted.
<i>MSG_RESET</i>	if the mailbox has been reset.
<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:



6.6.3.9 msg_t chMBPostAheadI (mailbox_t * mbp, msg_t msg)

Posts an high priority message into a mailbox.

This variant is non-blocking, the function returns a timeout condition if the queue is full.

Parameters

in	<i>mbp</i>	the pointer to an initialized <code>mailbox_t</code> object
in	<i>msg</i>	the message to be posted on the mailbox

Returns

The operation status.

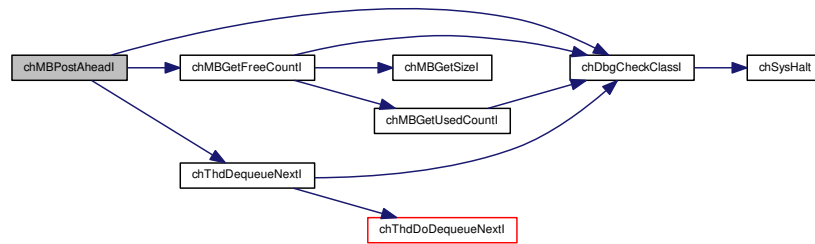
Return values

<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:



6.6.3.10 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 mailbox_t 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"> • <i>TIME_IMMEDIATE</i> immediate timeout. • <i>TIME_INFINITE</i> no timeout.

Returns

The operation status.

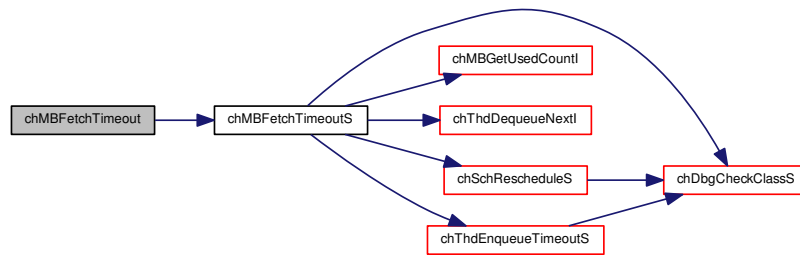
Return values

<i>MSG_OK</i>	if a message has been correctly 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:



6.6.3.11 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 mailbox_t 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"> • <i>TIME_IMMEDIATE</i> immediate timeout. • <i>TIME_INFINITE</i> no timeout.

Returns

The operation status.

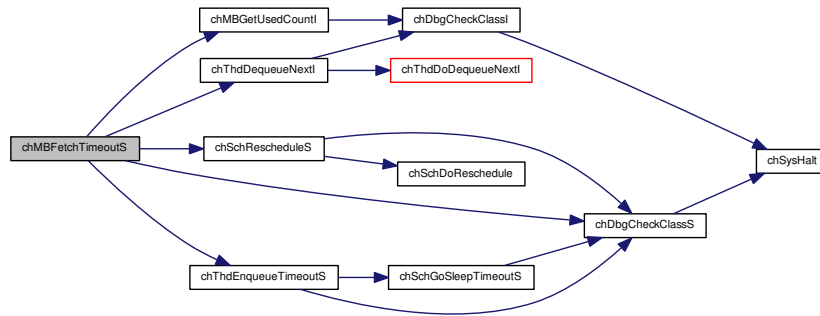
Return values

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



6.6.3.12 msg_t chMBFetchI (mailbox_t * mbp, msg_t * msgp)

Retrieves a message from a mailbox.

This variant is non-blocking, the function returns a timeout condition if the queue is empty.

Parameters

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.

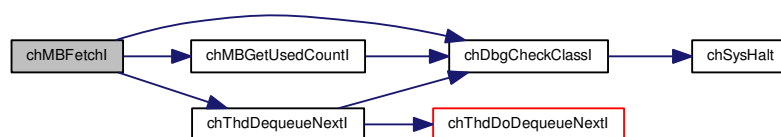
Return values

<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 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:



6.6.3.13 `static size_t chMBGetSizel (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.

6.6.3.14 `static size_t chMBGetUsedCountl (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 <code>mailbox_t</code> object
----	------------	---

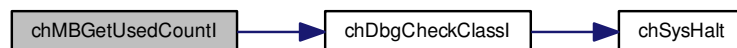
Returns

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:



6.6.3.15 `static size_t chMBGetFreeCountl (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 <code>mailbox_t</code> object
----	------------	---

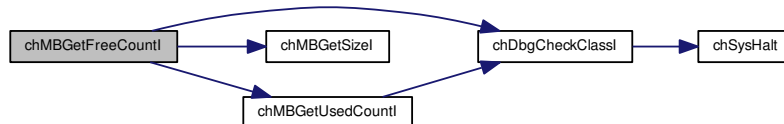
Returns

The number of empty message slots.

Function Class:

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

Here is the call graph for this function:



6.6.3.16 `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	<code>mbp</code>	the pointer to an initialized <code>mailbox_t</code> object
----	------------------	---

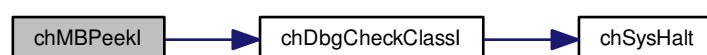
Returns

The next message in queue.

Function Class:

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

Here is the call graph for this function:



6.6.3.17 `static void chMBResumeX (mailbox_t * mbp)` `[inline],[static]`

Terminates the reset state.

Parameters

in	<i>mbp</i>	the pointer to an initialized mailbox_t object
----	------------	--

Function Class:

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

6.7 Memcore

6.7.1 Detailed Description

Core Memory Manager related APIs and services.

Operation mode

The core memory manager is a simplified allocator that only allows to allocate memory blocks without the possibility to free them.

This allocator is meant as a memory blocks provider for the other allocators such as:

- C-Runtime allocator (through a compiler specific adapter module).
- Heap allocator (see [Heaps](#)).
- Memory pools allocator (see [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.

Typedefs

- `typedef void (*)(size_t size, unsigned align)`
Memory get function.
- `typedef void (*)(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 * chCoreAllocAlignedWithOffset (size_t size, unsigned align, size_t offset)`
Allocates a memory block.

- void * [chCoreAllocAlignedWithOffset](#) (size_t size, unsigned align, size_t offset)
Allocates a memory block.
- size_t [chCoreGetStatusX](#) (void)
Core memory status.
- static void * [chCoreAllocAlignedl](#) (size_t size, unsigned align)
Allocates a memory block.
- static void * [chCoreAllocAligned](#) (size_t size, unsigned align)
Allocates a memory block.
- static void * [chCoreAlloccl](#) (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.

6.7.2 Macro Definition Documentation

6.7.2.1 #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**.

6.7.3 Typedef Documentation

6.7.3.1 typedef void*(* memgetfunc_t) (size_t size, unsigned align)

Memory get function.

6.7.3.2 typedef void*(* memgetfunc2_t) (size_t size, unsigned align, size_t offset)

Enhanced memory get function.

6.7.4 Function Documentation

6.7.4.1 void _core_init (void)

Low level memory manager initialization.

Function Class:

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

6.7.4.2 void * chCoreAllocAlignedWithOffsetI (size_t size, unsigned align, size_t offset)

Allocates a memory block.

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:

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:



6.7.4.3 void * chCoreAllocAlignedWithOffset (size_t size, unsigned align, size_t offset)

Allocates a memory block.

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.

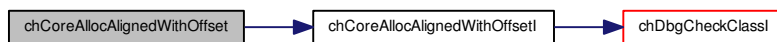
Return values

<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:

6.7.4.4 `size_t chCoreGetStatusX (void)`

Core memory status.

Returns

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

Function Class:

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

6.7.4.5 `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.

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.

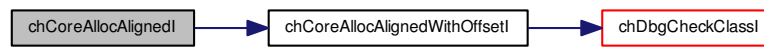
Return values

<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.

Here is the call graph for this function:



6.7.4.6 `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.

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.

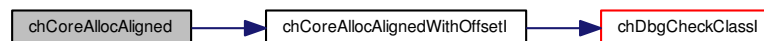
Return values

<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.

Here is the call graph for this function:



6.7.4.7 `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.

Parameters

in	<i>size</i>	the size of the block to be allocated.
----	-------------	--

Returns

A pointer to the allocated memory block.

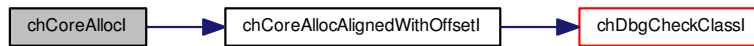
Return values

<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.

Here is the call graph for this function:



6.7.4.8 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.

Parameters

<i>in</i>	<i>size</i>	the size of the block to be allocated.
-----------	-------------	--

Returns

A pointer to the allocated memory block.

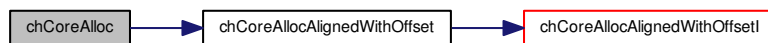
Return values

<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:



6.7.5 Variable Documentation

6.7.5.1 `memcore_t ch_memcore`

Memory core descriptor.

6.8 Pools

6.8.1 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.
- 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 [chPoolAddI](#) ([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 void [chGuardedPoolAdd](#) ([guarded_memory_pool_t](#) *gmp, void *objp)
Adds an object to a guarded memory pool.
- static void [chGuardedPoolAddI](#) ([guarded_memory_pool_t](#) *gmp, void *objp)
Adds an object to a guarded memory pool.
- static void * [chGuardedPoolAllocI](#) ([guarded_memory_pool_t](#) *gmp)
Allocates an object from a guarded memory pool.

6.8.2 Macro Definition Documentation

6.8.2.1 #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

6.8.2.2 `#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

6.8.2.3 `#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

6.8.2.4 `#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

6.8.3 Function Documentation

6.8.3.1 `void chPoolObjectInitAligned (memory_pool_t * mp, size_t size, unsigned align, memgetfunc_t provider)`

Initializes an empty memory pool.

Parameters

out	<i>mp</i>	pointer to a <code>memory_pool_t</code> structure
in	<i>size</i>	the size of the objects contained in this memory pool, the minimum accepted size is the size of a pointer to void.
in	<i>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

Function Class:

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

6.8.3.2 void chPoolLoadArray (memory_pool_t * mp, void * p, size_t n)

Loads a memory pool with an array of static objects.

Precondition

The memory pool must 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.

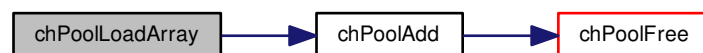
Parameters

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

**6.8.3.3 void * chPoolAlloc (memory_pool_t * mp)**

Allocates an object from a memory pool.

Precondition

The memory pool must already be initialized.

Parameters

in	<i>mp</i>	pointer to a <code>memory_pool_t</code> structure
----	-----------	---

Returns

The pointer to the allocated object.

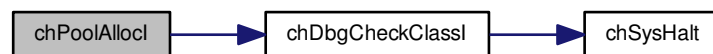
Return values

<code>NULL</code>	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:

**6.8.3.4 void * chPoolAlloc (memory_pool_t * mp)**

Allocates an object from a memory pool.

Precondition

The memory pool must already be initialized.

Parameters

in	<i>mp</i>	pointer to a <code>memory_pool_t</code> structure
----	-----------	---

Returns

The pointer to the allocated object.

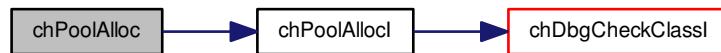
Return values

<code>NULL</code>	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:



6.8.3.5 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.

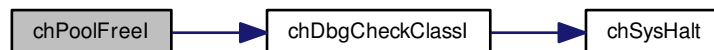
Parameters

in	<i>mp</i>	pointer to a memory_pool_t structure
in	<i>objp</i>	the pointer to the object to be released

Function Class:

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

Here is the call graph for this function:



6.8.3.6 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.

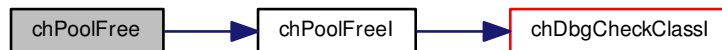
Parameters

in	<i>mp</i>	pointer to a memory_pool_t 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:



6.8.3.7 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 guarded_memory_pool_t 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:



6.8.3.8 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.

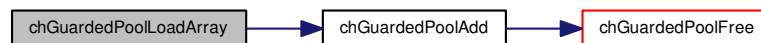
Parameters

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:



6.8.3.9 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 guarded_memory_pool_t 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"> • <i>TIME_IMMEDIATE</i> immediate timeout. • <i>TIME_INFINITE</i> no timeout.

Returns

The pointer to the allocated object.

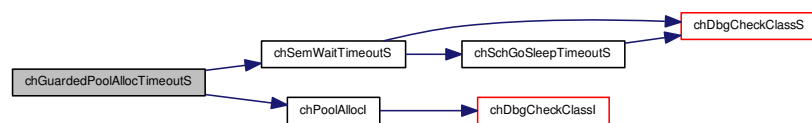
Return values

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

Function Class:

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

Here is the call graph for this function:



6.8.3.10 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 guarded_memory_pool_t 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"> • <i>TIME_IMMEDIATE</i> immediate timeout. • <i>TIME_INFINITE</i> no timeout.

Returns

The pointer to the allocated object.

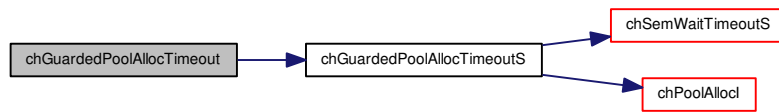
Return values

<i>NULL</i>	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:



6.8.3.11 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.

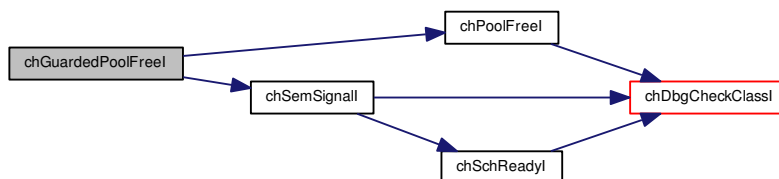
Parameters

in	<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 **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:



6.8.3.12 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.

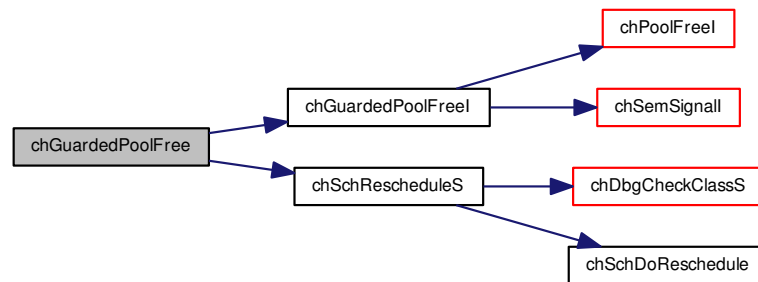
Parameters

in	<i>gmp</i>	pointer to a guarded_memory_pool_t 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:



6.8.3.13 `static void chPoolObjectInit (memory_pool_t * mp, size_t size, memgetfunc_t provider) [inline], [static]`

Initializes an empty memory pool.

Parameters

out	<i>mp</i>	pointer to a memory_pool_t structure
in	<i>size</i>	the size of the objects contained in this memory pool, the minimum accepted size is the size of a pointer to void.
in	<i>provider</i>	memory provider function for the memory pool or <code>NULL</code> if the pool is not allowed to grow automatically

Function Class:

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

Here is the call graph for this function:



6.8.3.14 `static void chPoolAdd (memory_pool_t * mp, void * objp)` `[inline], [static]`

Adds an object to a memory pool.

Precondition

The memory pool must be already been initialized.
 The added object must be of the right size for the specified memory pool.
 The added object must be memory aligned to the size of `stkalign_t` type.

Note

This function is just an alias for `chPoolFree()` and has been added for clarity.

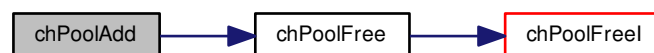
Parameters

in	<i>mp</i>	pointer to a <code>memory_pool_t</code> structure
in	<i>objp</i>	the pointer to the object to be added

Function Class:

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

Here is the call graph for this function:



6.8.3.15 `static void chPoolAddL (memory_pool_t * mp, void * objp)` `[inline], [static]`

Adds an object to a memory pool.

Precondition

The memory pool must be already been initialized.
 The added object must be of the right size for the specified memory pool.
 The added object must be memory aligned to the size of `stkalign_t` type.

Note

This function is just an alias for `chPoolFreeI()` and has been added for clarity.

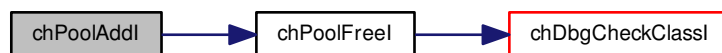
Parameters

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:



6.8.3.16 `static void chGuardedPoolObjectInit (guarded_memory_pool_t * gmp, size_t size) [inline], [static]`

Initializes an empty guarded memory pool.

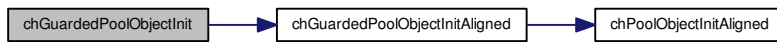
Parameters

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:



6.8.3.17 `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.

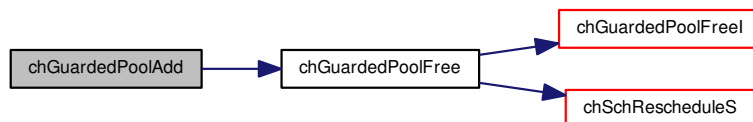
Parameters

in	<i>gmp</i>	pointer to a <code>guarded_memory_pool_t</code> structure
in	<i>objp</i>	the pointer to the object to be added

Function Class:

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

Here is the call graph for this function:



6.8.3.18 `static void chGuardedPoolAddl (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.

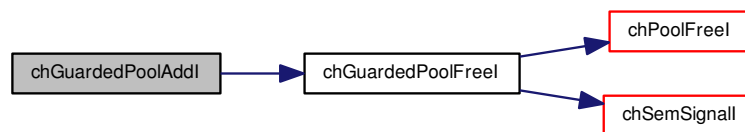
Parameters

in	<i>gmp</i>	pointer to a <code>guarded_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:



6.8.3.19 `static void* chGuardedPoolAlloc (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 <code>guarded_memory_pool_t</code> structure
----	------------	---

Returns

The pointer to the allocated object.

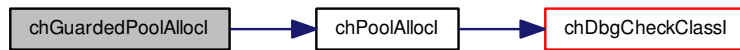
Return values

<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:



6.9 Binary_semaphores

6.9.1 Detailed Description

Binary semaphores related APIs and services.

Operation mode

Binary semaphores are implemented as a set of inline functions that use the existing counting semaphores primitives. The difference between counting and binary semaphores is that the counter of binary semaphores is not allowed to grow above the value 1. Repeated signal operation are ignored. A binary semaphore can thus have only two defined states:

- **Taken**, when its counter has a value of zero or lower than zero. A negative number represent the number of threads queued on the binary semaphore.
- **Not taken**, when its counter has a value of one.

Binary semaphores are different from mutexes because there is no concept of ownership, a binary semaphore can be taken by a thread and signaled by another thread or an interrupt handler, mutexes can only be taken and released by the same thread. Another difference is that binary semaphores, unlike mutexes, do not implement the priority inheritance protocol.

In order to use the binary semaphores APIs the `CH_CFG_USE_SEMAPHORES` option must be enabled in `chconf.h`.

Macros

- `#define _BSEMAPHORE_DATA(name, taken) {_SEMAPHORE_DATA(name.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 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 **chBSemResetI** (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 **chBSemSignalI** (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.

6.9.2 Macro Definition Documentation

6.9.2.1 **#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

6.9.2.2 **#define BSEMAPHORE_DECL(name, taken) binary_semaphore_t name = _BSEMAPHORE_DATA(name, taken)**

Static semaphore initializer.

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

Parameters

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

6.9.3 Typedef Documentation

6.9.3.1 **typedef struct ch_binary_semaphore binary_semaphore_t**

Binary semaphore type.

6.9.4 Function Documentation

6.9.4.1 **static void chBSemObjectInit (binary_semaphore_t * bsp, bool taken)** [inline], [static]

Initializes a binary semaphore.

Parameters

out	<i>bsp</i>	pointer to a <code>binary_semaphore_t</code> structure
in	<i>taken</i>	initial state of the binary semaphore: <ul style="list-style-type: none"> • <i>false</i>, the initial state is not taken. • <i>true</i>, the initial state is taken.

Function Class:

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

6.9.4.2 `static msg_t chBSemWait (binary_semaphore_t * bsp) [inline],[static]`

Wait operation on the binary semaphore.

Parameters

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

6.9.4.3 `static msg_t chBSemWaitS (binary_semaphore_t * bsp) [inline],[static]`

Wait operation on the binary semaphore.

Parameters

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

Returns

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

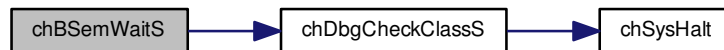
Return values

<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:



6.9.4.4 `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.

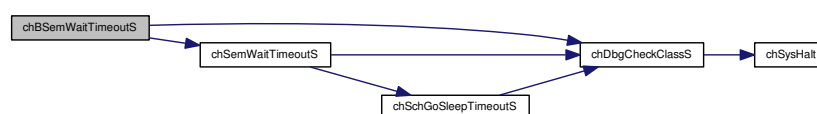
Return values

<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:



6.9.4.5 static msg_t chBSemWaitTimeout (binary_semaphore_t * bsp, sysinterval_t timeout) [inline],
[static]

Wait operation on the binary semaphore.

Parameters

in	bsp	pointer to a binary_semaphore_t structure
in	timeout	the number of ticks before the operation timeouts, the following special values are allowed: <ul style="list-style-type: none"> • <i>TIME_IMMEDIATE</i> immediate timeout. • <i>TIME_INFINITE</i> no timeout.

Returns

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

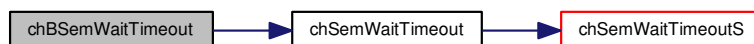
Return values

<i>MSG_OK</i>	if the binary semaphore has been successfully taken.
<i>MSG_RESET</i>	if the binary semaphore has been reset using bsemReset().
<i>MSG_TIMEOUT</i>	if the binary semaphore has not been signaled or reset within the specified timeout.

Function Class:

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

Here is the call graph for this function:



6.9.4.6 static void chBSemReset! (binary_semaphore_t * bsp, bool taken) [inline], [static]

Reset operation on the binary semaphore.

Note

The released threads can recognize they were waked up by a reset rather than a signal because the bsem←
Wait() will return MSG_RESET instead of MSG_OK.
This function does not reschedule.

Parameters

in	bsp	pointer to a binary_semaphore_t structure
----	-----	---

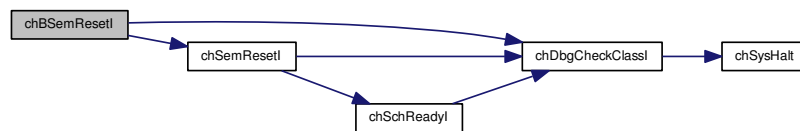
Parameters

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:



6.9.4.7 static void chBSemReset (binary_semaphore_t * bsp, bool taken) [inline], [static]

Reset operation on the binary semaphore.

Note

The released threads can recognize they were waked up by a reset rather than a signal because the `bsem←Wait()` will return `MSG_RESET` instead of `MSG_OK`.

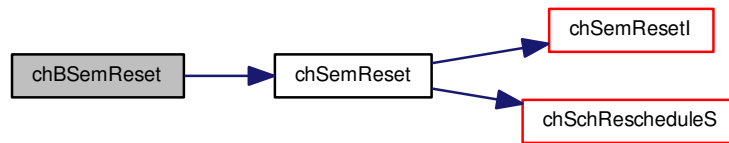
Parameters

in	<i>bsp</i>	pointer to a <code>binary_semaphore_t</code> structure
in	<i>taken</i>	new state of the binary semaphore <ul style="list-style-type: none"> <i>false</i>, the new state is not taken. <i>true</i>, the new state is taken.

Function Class:

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

Here is the call graph for this function:



6.9.4.8 `static void chBSemSignal (binary_semaphore_t * bsp) [inline],[static]`

Performs a signal operation on a binary semaphore.

Note

This function does not reschedule.

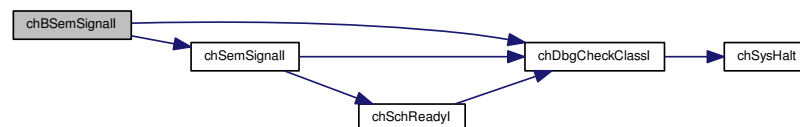
Parameters

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:



6.9.4.9 `static void chBSemSignal (binary_semaphore_t * bsp) [inline],[static]`

Performs a signal operation on a binary semaphore.

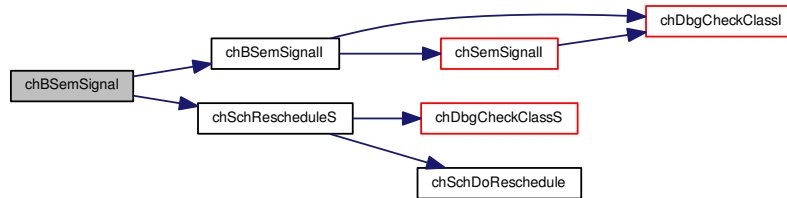
Parameters

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



6.9.4.10 `static bool chBSemGetStatel (const binary_semaphore_t * bsp) [inline],[static]`

Returns the binary semaphore current state.

Parameters

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

Returns

The binary semaphore current state.

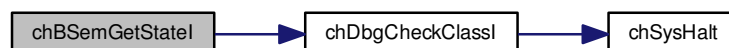
Return values

<i>false</i>	if the binary semaphore is not taken.
<i>true</i>	if the binary semaphore is taken.

Function Class:

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

Here is the call graph for this function:



6.10 Objects_fifo

6.10.1 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 [chFifoObjectInit](#) ([objects_fifo_t](#) *ofp, size_t objsize, size_t objn, unsigned objalign, 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 [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 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.

6.10.2 Typedef Documentation

6.10.2.1 typedef struct [ch_objects_fifo](#) [objects_fifo_t](#)

Type of an objects FIFO.

6.10.3 Function Documentation

6.10.3.1 `static void chFifoObjectInit (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.

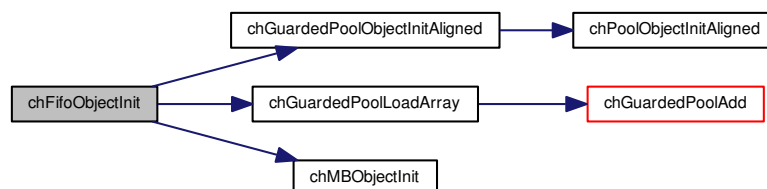
Parameters

out	<i>ofp</i>	pointer to a <code>objects_fifo_t</code> 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 <code>objn</code> objects of <code>objsize</code> size with <code>objalign</code> alignment
in	<i>msgbuf</i>	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:



6.10.3.2 `static void* chFifoTakeObjectI (objects_fifo_t * ofp)` [inline],[static]

Allocates a free object.

Parameters

in	<i>ofp</i>	pointer to a <code>objects_fifo_t</code> structure
----	------------	--

Returns

The pointer to the allocated object.

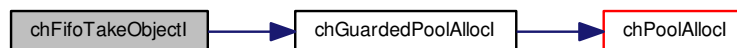
Return values

<code>NULL</code>	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:



6.10.3.3 `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.

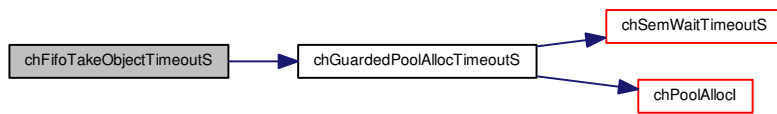
Return values

<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:



6.10.3.4 `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.

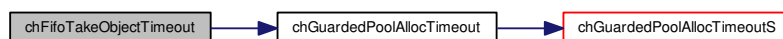
Return values

<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:



6.10.3.5 `static void chFifoReturnObject! (objects_fifo_t * ofp, void * objp) [inline], [static]`

Releases a fetched object.

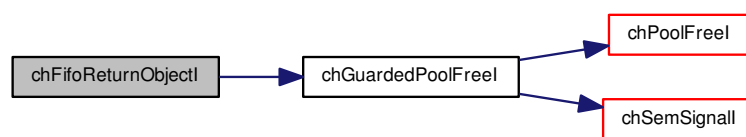
Parameters

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:



6.10.3.6 `static void chFifoReturnObject (objects_fifo_t * ofp, void * objp) [inline], [static]`

Releases a fetched object.

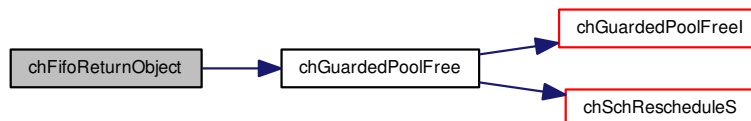
Parameters

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:

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

Here is the call graph for this function:



6.10.3.7 `static void chFifoSendObjectI (objects_fifo_t * ofp, void * objp) [inline], [static]`

Posts an object.

Note

By design the object can be always immediately posted.

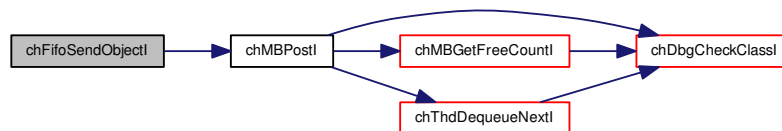
Parameters

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:



6.10.3.8 `static void chFifoSendObjectS (objects_fifo_t * ofp, void * objp) [inline], [static]`

Posts an object.

Note

By design the object can be always immediately posted.

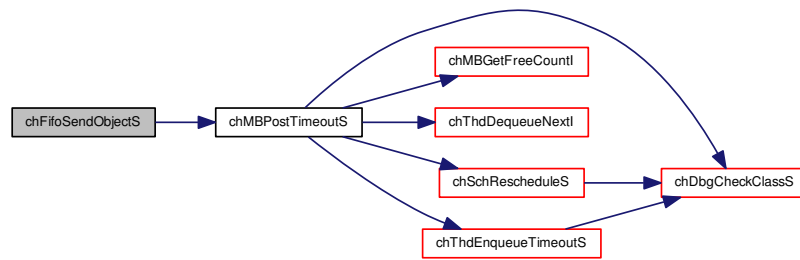
Parameters

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:



6.10.3.9 `static void chFifoSendObject (objects_fifo_t * ofp, void * objp)` [inline],[static]

Posts an object.

Note

By design the object can be always immediately posted.

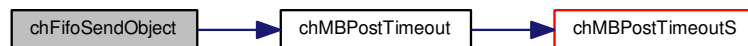
Parameters

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:



6.10.3.10 `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.

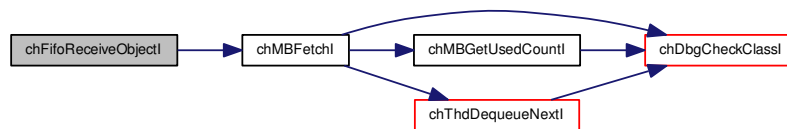
Return values

<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:



6.10.3.11 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 objects_fifo_t 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"> • <i>TIME_IMMEDIATE</i> immediate timeout. • <i>TIME_INFINITE</i> no timeout.

Returns

The operation status.

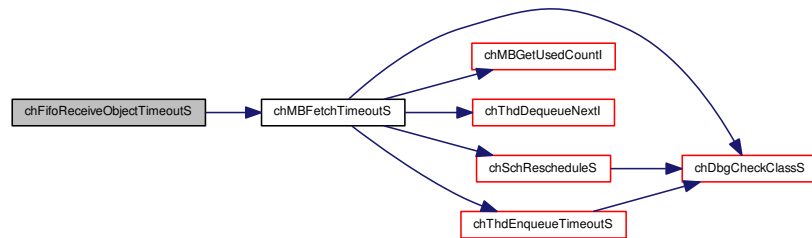
Return values

<i>MSG_OK</i>	if an object has been correctly fetched.
<i>MSG_TIMEOUT</i>	if the operation has timed out.

Function Class:

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:



6.10.3.12 static msg_t chFifoReceiveObjectTimeout (objects_fifo_t * ofp, void ** objpp, sysinterval_t timeout)
[inline],[static]

Fetches an object.

Parameters

in	ofp	pointer to a objects_fifo_t structure
in	objpp	pointer to the fetched object reference
in	timeout	the number of ticks before the operation timeouts, the following special values are allowed: <ul style="list-style-type: none"> • <i>TIME_IMMEDIATE</i> immediate timeout. • <i>TIME_INFINITE</i> no timeout.

Returns

The operation status.

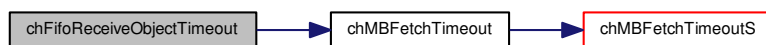
Return values

<i>MSG_OK</i>	if 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:



Chapter 7

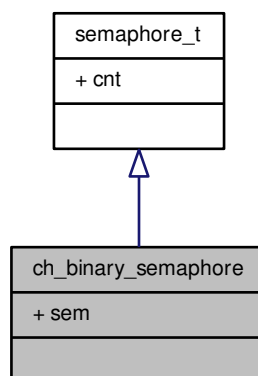
Data Structure Documentation

7.1 ch_binary_semaphore Struct Reference

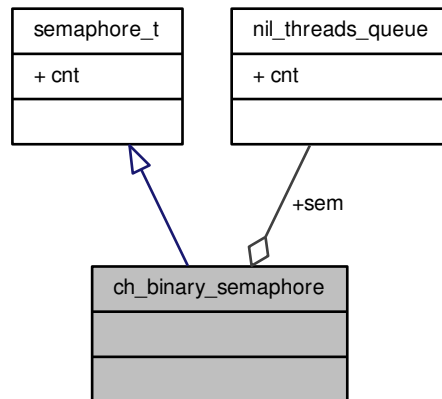
Binary semaphore type.

```
#include <chbsem.h>
```

Inheritance diagram for ch_binary_semaphore:



Collaboration diagram for `ch_binary_semaphore`:



Additional Inherited Members

7.1.1 Detailed Description

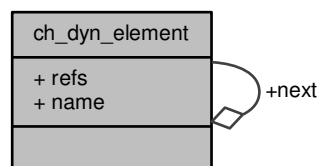
Binary semaphore type.

7.2 `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.

7.2.1 Detailed Description

Type of a dynamic object list element.

7.2.2 Field Documentation

7.2.2.1 `struct ch_dyn_element* ch_dyn_element::next`

Next dynamic object in the list.

7.2.2.2 `ucnt_t ch_dyn_element::refs`

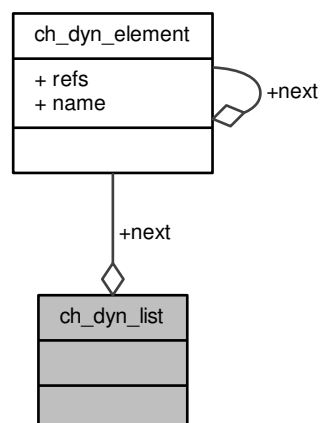
Number of references to this object.

7.3 ch_dyn_list Struct Reference

Type of a dynamic object list.

```
#include <chfactory.h>
```

Collaboration diagram for `ch_dyn_list`:



7.3.1 Detailed Description

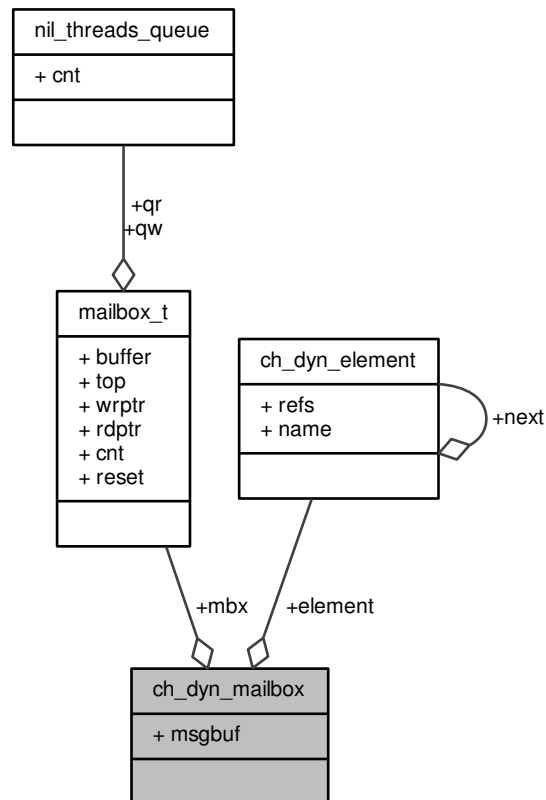
Type of a dynamic object list.

7.4 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.
- `msg_t` [msgbuf](#) []
Messages buffer.

7.4.1 Detailed Description

Type of a dynamic buffer object.

7.4.2 Field Documentation

7.4.2.1 dyn_element_t ch_dyn_mailbox::element

List element of the dynamic buffer object.

7.4.2.2 mailbox_t ch_dyn_mailbox::mbx

The mailbox.

7.4.2.3 msg_t ch_dyn_mailbox::msgbuf[]

Messages buffer.

Note

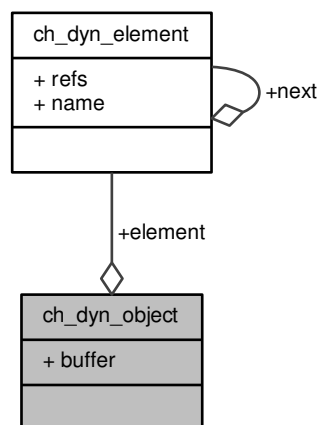
This requires C99.

7.5 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.
- [uint8_t buffer \[\]](#)
The buffer.

7.5.1 Detailed Description

Type of a dynamic buffer object.

7.5.2 Field Documentation

7.5.2.1 `dyn_element_t ch_dyn_object::element`

List element of the dynamic buffer object.

7.5.2.2 `uint8_t ch_dyn_object::buffer[]`

The buffer.

Note

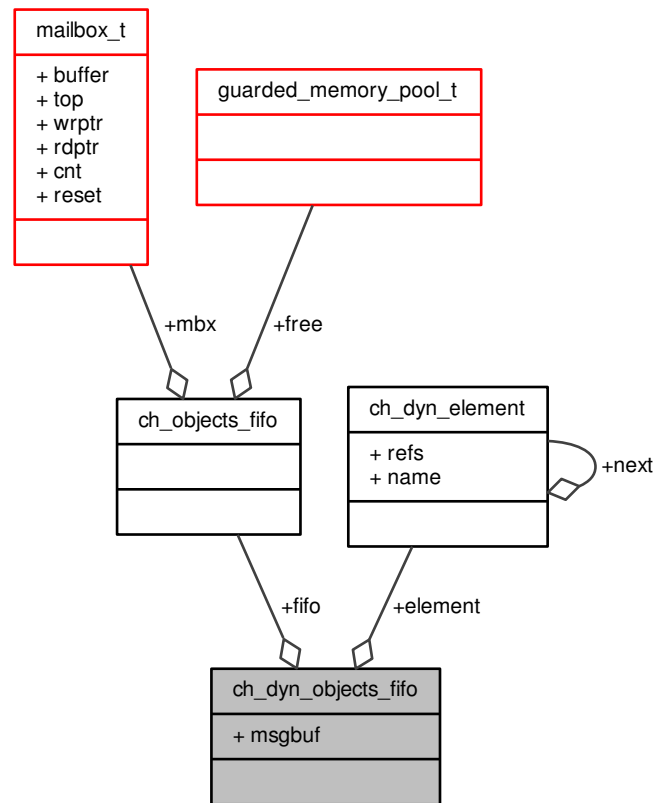
This requires C99.

7.6 `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.
- `msg_t msgbuf []`
Messages buffer.

7.6.1 Detailed Description

Type of a dynamic buffer object.

7.6.2 Field Documentation

7.6.2.1 `dyn_element_t ch_dyn_objects_fifo::element`

List element of the dynamic buffer object.

7.6.2.2 `objects_fifo_t ch_dyn_objects_fifo::fifo`

The objects FIFO.

7.6.2.3 `msg_t ch_dyn_objects_fifo::msgbuf[]`

Messages buffer.

Note

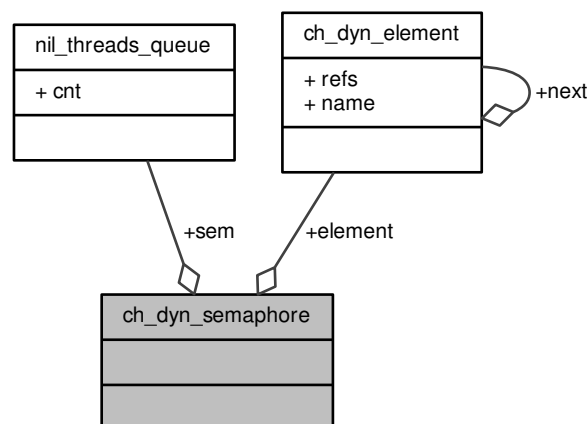
This open array is followed by another area containing the objects, this area is not represented in this structure. This requires C99.

7.7 `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.

7.7.1 Detailed Description

Type of a dynamic semaphore.

7.7.2 Field Documentation

7.7.2.1 dyn_element_t ch_dyn_semaphore::element

List element of the dynamic semaphore.

7.7.2.2 semaphore_t ch_dyn_semaphore::sem

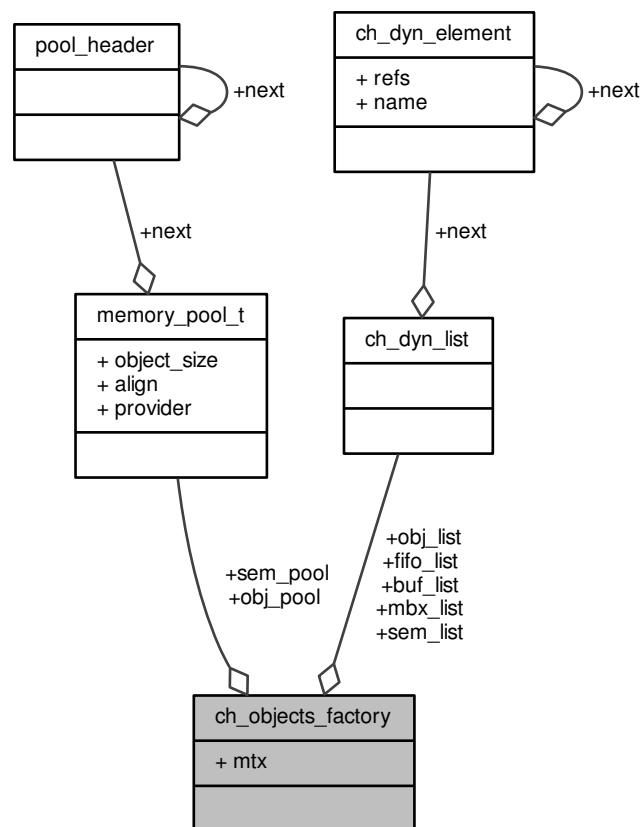
The semaphore.

7.8 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.

7.8.1 Detailed Description

Type of the factory main object.

7.8.2 Field Documentation

7.8.2.1 `mutex_t ch_objects_factory::mtx`

Factory access mutex or semaphore.

7.8.2.2 `dyn_list_t ch_objects_factory::obj_list`

List of the registered objects.

7.8.2.3 `memory_pool_t ch_objects_factory::obj_pool`

Pool of the available registered objects.

7.8.2.4 `dyn_list_t ch_objects_factory::buf_list`

List of the allocated buffer objects.

7.8.2.5 `dyn_list_t ch_objects_factory::sem_list`

List of the allocated semaphores.

7.8.2.6 `memory_pool_t ch_objects_factory::sem_pool`

Pool of the available semaphores.

7.8.2.7 `dyn_list_t ch_objects_factory::mbx_list`

List of the allocated buffer objects.

7.8.2.8 dyn_list_t ch_objects_factory::fifo_list

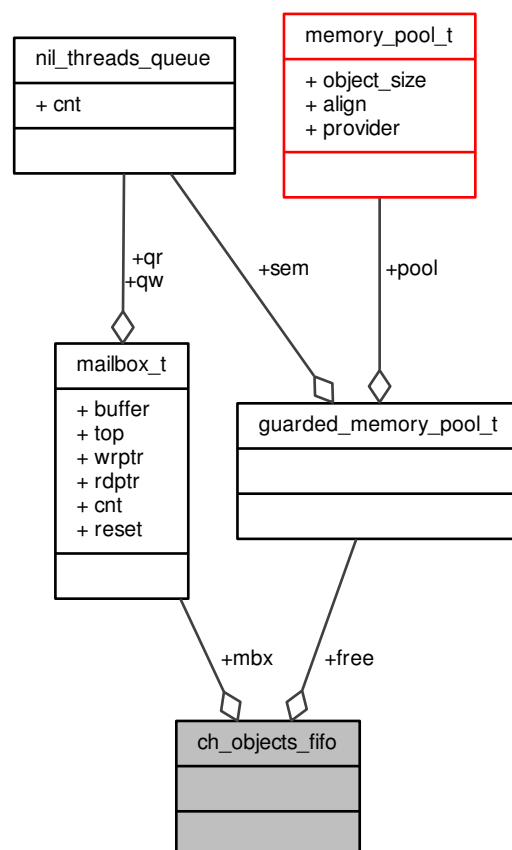
List of the allocated "objects FIFO" objects.

7.9 ch_objects_fifo Struct Reference

Type of an objects FIFO.

```
#include <chfifo.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.

7.9.1 Detailed Description

Type of an objects FIFO.

7.9.2 Field Documentation

7.9.2.1 `guarded_memory_pool_t ch_objects_fifo::free`

Pool of the free objects.

7.9.2.2 `mailbox_t ch_objects_fifo::mbx`

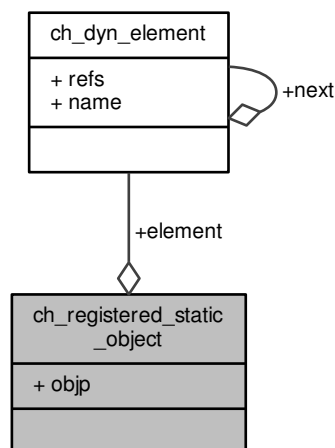
Mailbox of the sent objects.

7.10 `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.

7.10.1 Detailed Description

Type of a registered object.

7.10.2 Field Documentation

7.10.2.1 dyn_element_t ch_registered_static_object::element

List element of the registered object.

7.10.2.2 void* ch_registered_static_object::objp

Pointer to the object.

Note

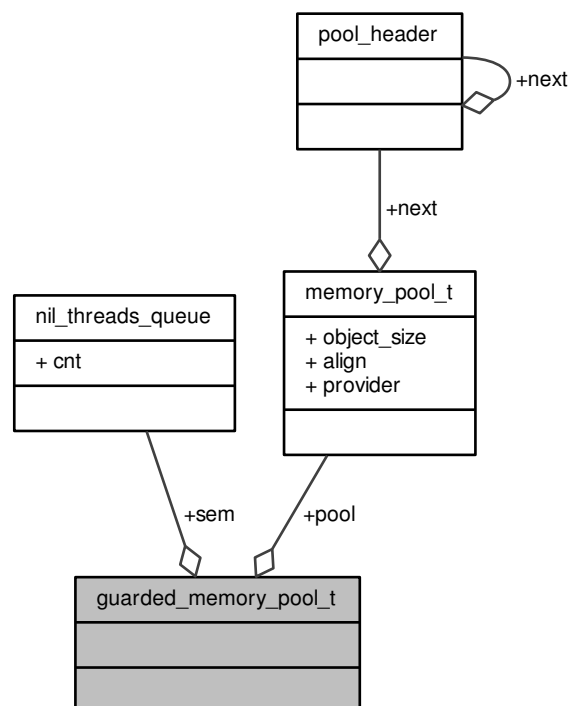
The type of the object is not stored in anyway.

7.11 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.

7.11.1 Detailed Description

Guarded memory pool descriptor.

7.11.2 Field Documentation

7.11.2.1 semaphore_t guarded_memory_pool_t::sem

Counter semaphore guarding the memory pool.

7.11.2.2 memory_pool_t guarded_memory_pool_t::pool

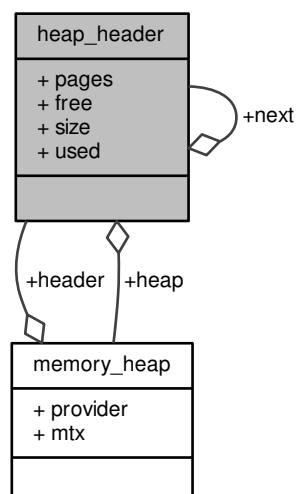
The memory pool itself.

7.12 heap_header Union Reference

Memory heap block header.

```
#include <chheap.h>
```

Collaboration diagram for heap_header:



7.12.1 Detailed Description

Memory heap block header.

7.12.2 Field Documentation

7.12.2.1 heap_header_t* heap_header::next

Next block in free list.

7.12.2.2 size_t heap_header::pages

Size of the area in pages.

7.12.2.3 memory_heap_t* heap_header::heap

Block owner heap.

7.12.2.4 size_t heap_header::size

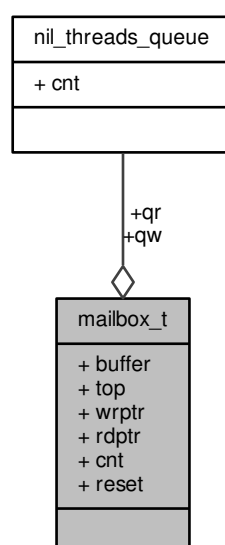
Size of the area in bytes.

7.13 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.

7.13.1 Detailed Description

Structure representing a mailbox object.

7.13.2 Field Documentation

7.13.2.1 `msg_t* mailbox_t::buffer`

Pointer to the mailbox buffer.

7.13.2.2 `msg_t* mailbox_t::top`

Pointer to the location after the buffer.

7.13.2.3 `msg_t* mailbox_t::wrptr`

Write pointer.

7.13.2.4 `msg_t* mailbox_t::rdptr`

Read pointer.

7.13.2.5 `size_t mailbox_t::cnt`

Messages in queue.

7.13.2.6 `bool mailbox_t::reset`

True in reset state.

7.13.2.7 threads_queue_t mailbox_t::qw

Queued writers.

7.13.2.8 threads_queue_t mailbox_t::qr

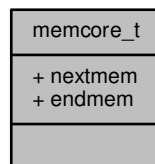
Queued readers.

7.14 memcore_t Struct Reference

Type of memory core object.

```
#include <chmemcore.h>
```

Collaboration diagram for memcore_t:



Data Fields

- `uint8_t * nextmem`
Next free address.
- `uint8_t * endmem`
Final address.

7.14.1 Detailed Description

Type of memory core object.

7.14.2 Field Documentation

7.14.2.1 uint8_t* memcore_t::nextmem

Next free address.

7.14.2.2 uint8_t* memcore_t::endmem

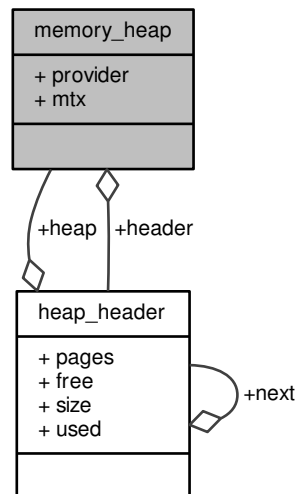
Final address.

7.15 memory_heap Struct Reference

Structure describing a memory heap.

```
#include <chheap.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.

7.15.1 Detailed Description

Structure describing a memory heap.

7.15.2 Field Documentation

7.15.2.1 memgetfunc2_t memory_heap::provider

Memory blocks provider for this heap.

7.15.2.2 heap_header_t memory_heap::header

Free blocks list header.

7.15.2.3 mutex_t memory_heap::mtx

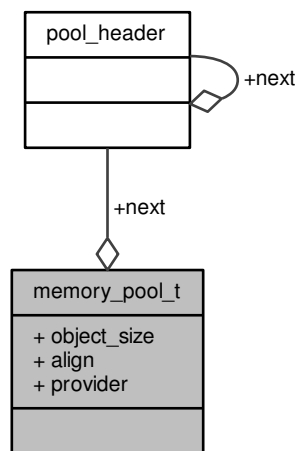
Heap access mutex.

7.16 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.

7.16.1 Detailed Description

Memory pool descriptor.

7.16.2 Field Documentation

7.16.2.1 struct [pool_header](#)* [memory_pool_t::next](#)

Pointer to the header.

7.16.2.2 `size_t memory_pool_t::object_size`

Memory pool objects size.

7.16.2.3 `unsigned memory_pool_t::align`

Required alignment.

7.16.2.4 `memgetfunc_t memory_pool_t::provider`

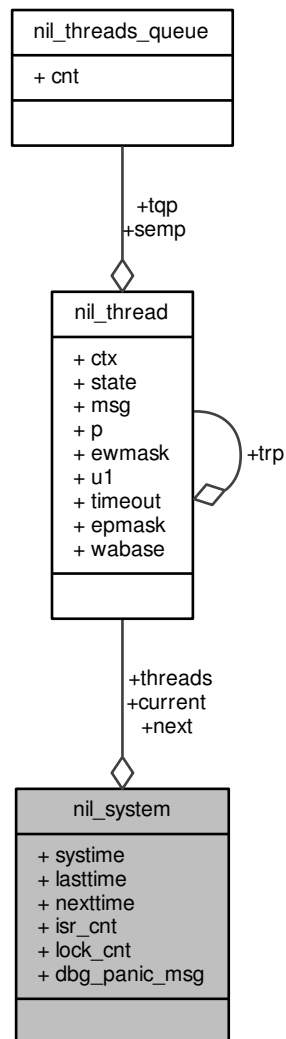
Memory blocks provider for this pool.

7.17 `nil_system` Struct Reference

System data structure.

```
#include <ch.h>
```

Collaboration diagram for nil_system:



Data Fields

- `thread_t * current`
Pointer to the running thread.
- `thread_t * next`
Pointer to the next thread to be executed.
- volatile `systime_t systime`
System time.
- `systime_t lasttime`
System time of the last tick event.
- `systime_t nexttime`
Time of the next scheduled tick event.
- `cnt_t isr_cnt`

ISR nesting level.

- `cnt_t lock_cnt`

Lock nesting level.

- `const char *volatile dbg_panic_msg`

Panic message.

- `thread_t threads [CH_CFG_NUM_THREADS+1]`

Thread structures for all the defined threads.

7.17.1 Detailed Description

System data structure.

Note

This structure contain all the data areas used by the OS except stacks.

7.17.2 Field Documentation

7.17.2.1 `thread_t* nil_system::current`

Pointer to the running thread.

7.17.2.2 `thread_t* nil_system::next`

Pointer to the next thread to be executed.

Note

This pointer must point at the same thread pointed by `current` or to an higher priority thread if a switch is required.

7.17.2.3 `volatile systime_t nil_system::systime`

System time.

7.17.2.4 `systime_t nil_system::lasttime`

System time of the last tick event.

7.17.2.5 `systime_t nil_system::nexttime`

Time of the next scheduled tick event.

7.17.2.6 `cnt_t nil_system::isr_cnt`

ISR nesting level.

7.17.2.7 `cnt_t nil_system::lock_cnt`

Lock nesting level.

7.17.2.8 `const char* volatile nil_system::dbg_panic_msg`

Panic message.

Note

This field is only present if some debug options have been activated.
 Accesses to this pointer must never be optimized out so the field itself is declared volatile.

7.17.2.9 `thread_t nil_system::threads[CH_CFG_NUM_THREADS+1]`

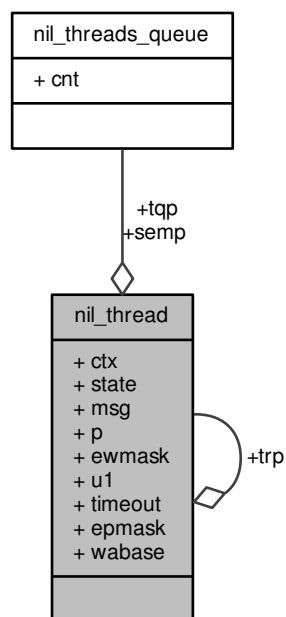
Thread structures for all the defined threads.

7.18 nil_thread Struct Reference

Structure representing a thread.

```
#include <ch.h>
```

Collaboration diagram for nil_thread:



Data Fields

- struct port_context [ctx](#)
Processor context.
- tstate_t [state](#)
Thread state.

- volatile `sysinterval_t` `timeout`
Timeout counter, zero if disabled.
- `eventmask_t` `epmask`
Pending events mask.
- `stkalign_t` * `wabase`
Thread stack boundary.
- `msg_t` `msg`
Wake-up message.
- `void` * `p`
Generic pointer.
- `thread_reference_t` * `trp`
Pointer to thread reference.
- `threads_queue_t` * `tqp`
Pointer to thread queue.
- `semaphore_t` * `semp`
Pointer to semaphore.
- `eventmask_t` `ewmask`
Enabled events mask.

7.18.1 Detailed Description

Structure representing a thread.

7.18.2 Field Documentation

7.18.2.1 `struct port_context` `nil_thread::ctx`

Processor context.

7.18.2.2 `tstate_t` `nil_thread::state`

Thread state.

7.18.2.3 `msg_t` `nil_thread::msg`

Wake-up message.

7.18.2.4 `void*` `nil_thread::p`

Generic pointer.

7.18.2.5 `thread_reference_t*` `nil_thread::trp`

Pointer to thread reference.

7.18.2.6 `threads_queue_t*` `nil_thread::tqp`

Pointer to thread queue.

7.18.2.7 semaphore_t* nil_thread::semp

Pointer to semaphore.

7.18.2.8 eventmask_t nil_thread::ewmask

Enabled events mask.

7.18.2.9 volatile sysinterval_t nil_thread::timeout

Timeout counter, zero if disabled.

7.18.2.10 eventmask_t nil_thread::epmask

Pending events mask.

7.18.2.11 stkalign_t* nil_thread::wabase

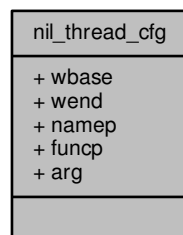
Thread stack boundary.

7.19 nil_thread_cfg Struct Reference

Structure representing a thread static configuration.

```
#include <ch.h>
```

Collaboration diagram for nil_thread_cfg:

**Data Fields**

- `stkalign_t * wbase`
Thread working area base.
- `stkalign_t * wend`
Thread working area end.
- `const char * namep`
Thread name, for debugging.
- `tfunc_t funcp`

Thread function.

- void * [arg](#)

Thread function argument.

7.19.1 Detailed Description

Structure representing a thread static configuration.

7.19.2 Field Documentation

7.19.2.1 `stkaligned_t* nil_thread_cfg::wbase`

Thread working area base.

7.19.2.2 `stkaligned_t* nil_thread_cfg::wend`

Thread working area end.

7.19.2.3 `const char* nil_thread_cfg::namep`

Thread name, for debugging.

7.19.2.4 `tfunc_t nil_thread_cfg::funcp`

Thread function.

7.19.2.5 `void* nil_thread_cfg::arg`

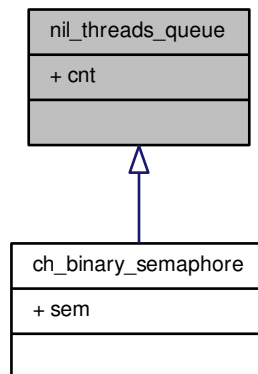
Thread function argument.

7.20 `nil_threads_queue` Struct Reference

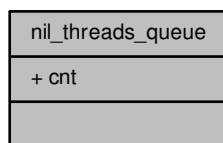
Structure representing a queue of threads.

```
#include <ch.h>
```

Inheritance diagram for nil_threads_queue:



Collaboration diagram for nil_threads_queue:



Data Fields

- volatile cnt_t [cnt](#)

Threads Queue counter.

7.20.1 Detailed Description

Structure representing a queue of threads.

7.20.2 Field Documentation

7.20.2.1 volatile cnt_t nil_threads_queue::cnt

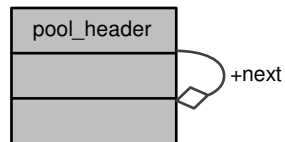
Threads Queue counter.

7.21 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.

7.21.1 Detailed Description

Memory pool free object header.

7.21.2 Field Documentation

7.21.2.1 struct `pool_header`* `pool_header::next`

Pointer to the next pool header in the list.

Chapter 8

File Documentation

8.1 ch.c File Reference

Nil RTOS main source file.

```
#include "ch.h"
```

Functions

- void [_dbg_check_disable](#) (void)
Guard code for [chSysDisable\(\)](#).
- void [_dbg_check_suspend](#) (void)
Guard code for [chSysSuspend\(\)](#).
- void [_dbg_check_enable](#) (void)
Guard code for [chSysEnable\(\)](#).
- void [_dbg_check_lock](#) (void)
Guard code for [chSysLock\(\)](#).
- void [_dbg_check_unlock](#) (void)
Guard code for [chSysUnlock\(\)](#).
- void [_dbg_check_lock_from_isr](#) (void)
Guard code for [chSysLockFromIsr\(\)](#).
- void [_dbg_check_unlock_from_isr](#) (void)
Guard code for [chSysUnlockFromIsr\(\)](#).
- void [_dbg_check_enter_isr](#) (void)
Guard code for [CH_IRQ_PROLOGUE\(\)](#).
- void [_dbg_check_leave_isr](#) (void)
Guard code for [CH_IRQ_EPILOGUE\(\)](#).
- void [chDbgCheckClassI](#) (void)
I-class functions context check.
- void [chDbgCheckClassS](#) (void)
S-class functions context check.
- void [chSysInit](#) (void)
Initializes the kernel.
- void [chSysHalt](#) (const char *reason)
Halts the system.
- void [chSysTimerHandlerI](#) (void)
Time management handler.

- void [chSysUnconditionalLock](#) (void)
Unconditionally enters the kernel lock state.
- void [chSysUnconditionalUnlock](#) (void)
Unconditionally leaves the kernel lock state.
- 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.
- thread_t * [chSchReadyI](#) (thread_t *tp, msg_t msg)
Makes the specified thread ready for execution.
- bool [chSchIsPreemptionRequired](#) (void)
Evaluates if preemption is required.
- void [chSchDoReschedule](#) (void)
Switches to the first thread on the runnable queue.
- void [chSchRescheduleS](#) (void)
Reschedules if needed.
- msg_t [chSchGoSleepTimeoutS](#) (tstate_t newstate, sysinterval_t timeout)
Puts the current thread to sleep into the specified state with timeout specification.
- 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 [chThdSleep](#) (sysinterval_t timeout)
Suspends the invoking thread for the specified time.
- void [chThdSleepUntil](#) (systime_t abstime)
Suspends the invoking thread until the system time arrives to the specified value.
- msg_t [chThdEnqueueTimeoutS](#) (threads_queue_t *tqp, sysinterval_t timeout)
Enqueues the caller thread on a threads queue object.
- void [chThdDoDequeueNextI](#) (threads_queue_t *tqp, msg_t msg)
Dequeues and wakes up one thread from the 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.
- 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 [chSemReset](#) (semaphore_t *sp, cnt_t n)
Performs a reset operation on the semaphore.
- void [chSemResetI](#) (semaphore_t *sp, cnt_t n)
Performs a reset operation on the semaphore.
- void [chEvtSignal](#) (thread_t *tp, eventmask_t mask)

- Adds a set of event flags directly to the specified `thread_t`.*

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

Adds a set of event flags directly to the specified `thread_t`.
- `eventmask_t` `chEvtWaitAnyTimeout`(`eventmask_t` mask, `sysinterval_t` timeout)

Waits for any of the specified events.

Variables

- `nil_system_t` nil
- System data structures.*

8.1.1 Detailed Description

Nil RTOS main source file.

8.2 ch.h File Reference

Nil RTOS main header file.

```
#include "chtypes.h"
#include "chconf.h"
#include "chlicense.h"
#include "chcore.h"
#include "chmboxes.h"
#include "chmemcore.h"
#include "chheap.h"
#include "chmempools.h"
#include "chfifo.h"
#include "chfactory.h"
```

Data Structures

- struct `nil_threads_queue`
- Structure representing a queue of threads.*
- struct `nil_thread_cfg`
- Structure representing a thread static configuration.*
- struct `nil_thread`
- Structure representing a thread.*
- struct `nil_system`
- System data structure.*

Macros

- #define `_CHIBIOS_NIL_`
- ChibiOS/NIL identification macro.*
- #define `CH_KERNEL_STABLE` 1
- Stable release flag.*
- #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 `THD_IDLE_BASE` (`&__main_thread_stack_base__`)
- #define `__CH_STRINGIFY(a)` `#a`
Utility to make the parameter a quoted string.

ChibiOS/NIL version identification

- #define `CH_KERNEL_VERSION` `"3.0.0"`
Kernel version string.
- #define `CH_KERNEL_MAJOR` `3`
Kernel version major number.
- #define `CH_KERNEL_MINOR` `0`
Kernel version minor number.
- #define `CH_KERNEL_PATCH` `0`
Kernel version patch number.

Wakeup messages

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

Special time constants

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

Thread state related macros

- #define `NIL_STATE_READY` (`tstate_t`)`0`
Thread ready or executing.
- #define `NIL_STATE_SLEEPING` (`tstate_t`)`1`
Thread sleeping.
- #define `NIL_STATE_SUSP` (`tstate_t`)`2`
Thread suspended.
- #define `NIL_STATE_WTQUEUE` (`tstate_t`)`3`
On queue or semaph.

- #define `NIL_STATE_WTOREVT` (tstate_t)4
Waiting for events.
- #define `NIL_THD_IS_READY`(tr) ((tr)->state == `NIL_STATE_READY`)
- #define `NIL_THD_IS_SLEEPING`(tr) ((tr)->state == `NIL_STATE_SLEEPING`)
- #define `NIL_THD_IS_SUSP`(tr) ((tr)->state == `NIL_STATE_SUSP`)
- #define `NIL_THD_IS_WTQUEUE`(tr) ((tr)->state == `NIL_STATE_WTQUEUE`)
- #define `NIL_THD_IS_WTOREVT`(tr) ((tr)->state == `NIL_STATE_WTOREVT`)

Events related macros

- #define `ALL_EVENTS` ((eventmask_t)-1)
All events allowed mask.
- #define `EVENT_MASK`(eid) ((eventmask_t)(1 << (eid)))
Returns an event mask from an event identifier.

Threads tables definition macros

- #define `THD_TABLE_BEGIN` const `thread_config_t` nil_thd_configs[`CH_CFG_NUM_THREADS` + 1] = {
Start of user threads table.
- #define `THD_TABLE_ENTRY`(wap, name, funcp, arg)
Entry of user threads table.
- #define `THD_TABLE_END`
End of user threads table.

Memory alignment support macros

- #define `MEM_ALIGN_MASK`(a) ((size_t)(a) - 1U)
Alignment mask constant.
- #define `MEM_ALIGN_PREV`(p, a) ((size_t)(p) & ~`MEM_ALIGN_MASK`(a))
Aligns to the previous aligned memory address.
- #define `MEM_ALIGN_NEXT`(p, a)
Aligns to the new 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.

Working Areas

- #define `THD_WORKING_AREA_SIZE`(n)
Calculates the total Working Area size.
- #define `THD_WORKING_AREA`(s, n) `PORT_WORKING_AREA`(s, n)
Static working area allocation.

Threads abstraction macros

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

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

- #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.

Threads queues

- #define `_THREADS_QUEUE_DATA(name)` `{(cnt_t)0}`
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.

Semaphores macros

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

Macro Functions

- #define `chSysGetRealtimeCounterX()` `(rtcnt_t)port_rt_get_counter_value()`
Returns the current value of the system real time counter.
- #define `chSysDisable()`
Raises the system interrupt priority mask to the maximum level.
- #define `chSysSuspend()`
Raises the system interrupt priority mask to system level.
- #define `chSysEnable()`
Lowers the system interrupt priority mask to user level.
- #define `chSysLock()`
Enters the kernel lock state.
- #define `chSysUnlock()`
Leaves the kernel lock state.
- #define `chSysLockFromISR()`
Enters the kernel lock state from within an interrupt handler.
- #define `chSysUnlockFromISR()`
Leaves the kernel lock state from within an interrupt handler.

- `#define chSchIsRescRequiredI()` ((bool)(nil.current != nil.next))
Evaluates if a reschedule is required.
- `#define chThdGetSelfX()` nil.current
Returns a pointer to the current `thread_t`.
- `#define chThdSleepSeconds(secs) chThdSleep(TIME_S2I(secs))`
Delays the invoking thread for the specified number of seconds.
- `#define chThdSleepMilliseconds(msecs) chThdSleep(TIME_MS2I(msecs))`
Delays the invoking thread for the specified number of milliseconds.
- `#define chThdSleepMicroseconds(usecs) chThdSleep(TIME_US2I(usecs))`
Delays the invoking thread for the specified number of microseconds.
- `#define chThdSleepS(timeout) (void) chSchGoSleepTimeoutS(NIL_STATE_SLEEPING, timeout)`
Suspends the invoking thread for the specified time.
- `#define chThdSleepUntilS(abstime)`
Suspends the invoking thread until the system time arrives to the specified value.
- `#define chThdQueueObjectInit(tqp) ((tqp)->cnt = (cnt_t)0)`
Initializes a threads queue object.
- `#define chThdQueueIsEmptyI(tqp) ((bool)(tqp->cnt >= (cnt_t)0))`
Evaluates to `true` if the specified queue is empty.
- `#define chSemObjectInit(sp, n) ((sp)->cnt = n)`
Initializes a semaphore with the specified counter value.
- `#define chSemWait(sp) chSemWaitTimeout(sp, TIME_INFINITE)`
Performs a wait operation on a semaphore.
- `#define chSemWaitS(sp) chSemWaitTimeoutS(sp, TIME_INFINITE)`
Performs a wait operation on a semaphore.
- `#define chSemFastWaitI(sp) ((sp)->cnt--)`
Decreases the semaphore counter.
- `#define chSemFastSignalI(sp) ((sp)->cnt++)`
Increases the semaphore counter.
- `#define chSemGetCounterI(sp) ((sp)->cnt)`
Returns the semaphore counter current value.
- `#define chVTGetSystemTimeX()` (nil.systime)
Current system time.
- `#define chVTimeElapsedSinceX(start) chTimeDiffX((start), chVTGetSystemTimeX())`
Returns the elapsed time since the specified start time.
- `#define chTimeAddX(systime, interval) ((systime_t)(systime) + (systime_t)(interval))`
Adds an interval to a system time returning a system time.
- `#define chTimeDiffX(start, end) ((sysinterval_t)((systime_t)((systime_t)(end) - (systime_t)(start))))`
Subtracts two system times returning an interval.
- `#define chTimeIsInRangeX(time, start, end)`
Checks if the specified time is within the specified time range.
- `#define chDbgCheck(c)`
Function parameters check.
- `#define chDbgAssert(c, r)`
Condition assertion.

Typedefs

- `typedef uint32_t systime_t`
Type of system time.
- `typedef uint32_t sysinterval_t`
Type of time interval.
- `typedef uint64_t time_conv_t`
Type of time conversion variable.
- `typedef struct nil_thread thread_t`
Type of a structure representing a thread.
- `typedef struct nil_threads_queue threads_queue_t`

- Type of a queue of threads.*

 - typedef [threads_queue_t](#) semaphore_t

Type of a structure representing a semaphore.
- typedef void(* [tfunc_t](#)) (void *p)

Thread function.
- typedef struct [nil_thread_cfg](#) thread_config_t

Type of a structure representing a thread static configuration.
- typedef [thread_t](#) * [thread_reference_t](#)

Type of a thread reference.
- typedef struct [nil_system](#) nil_system_t

Type of a structure representing the system.

Functions

- void [chSysInit](#) (void)

Initializes the kernel.
- void [chSysHalt](#) (const char *reason)

Halts the system.
- void [chSysTimerHandlerI](#) (void)

Time management handler.
- void [chSysUnconditionalLock](#) (void)

Unconditionally enters the kernel lock state.
- void [chSysUnconditionalUnlock](#) (void)

Unconditionally leaves the kernel lock state.
- syssts_t [chSysGetStatusAndLockX](#) (void)

Returns the execution status and enters a critical zone.
- bool [chSysIsCounterWithinX](#) (rtcnt_t cnt, rctnt_t start, rctnt_t end)

Realtime window test.
- void [chSysPolledDelayX](#) (rtcnt_t cycles)

Polled delay.
- void [chSysRestoreStatusX](#) (syssts_t sts)

Restores the specified execution status and leaves a critical zone.
- [thread_t](#) * [chSchReadyI](#) ([thread_t](#) *tp, msg_t msg)

Makes the specified thread ready for execution.
- bool [chSchIsPreemptionRequired](#) (void)

Evaluates if preemption is required.
- void [chSchDoReschedule](#) (void)

Switches to the first thread on the runnable queue.
- void [chSchRescheduleS](#) (void)

Reschedules if needed.
- msg_t [chSchGoSleepTimeoutS](#) (tstate_t newstate, [sysinterval_t](#) timeout)

Puts the current thread to sleep into the specified state with timeout specification.
- 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 [chThdSleep](#) ([sysinterval_t](#) timeout)

Suspends the invoking thread for the specified time.
- void [chThdSleepUntil](#) ([systime_t](#) abstime)

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

- `msg_t chThdEnqueueTimeoutS (threads_queue_t *tqp, sysinterval_t timeout)`
Enqueues the caller thread on a threads queue object.
- `void chThdDoDequeueNextI (threads_queue_t *tqp, msg_t msg)`
Dequeues and wakes up one thread from the 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.
- `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 chSemSignalI (semaphore_t *sp)`
Performs a signal operation on a semaphore.
- `void chSemReset (semaphore_t *sp, cnt_t n)`
Performs a reset operation on the semaphore.
- `void chSemResetI (semaphore_t *sp, cnt_t n)`
Performs a reset operation on the semaphore.
- `void chEvtSignal (thread_t *tp, eventmask_t mask)`
Adds a set of event flags directly to the specified thread_t.
- `void chEvtSignalI (thread_t *tp, eventmask_t mask)`
Adds a set of event flags directly to the specified thread_t.
- `eventmask_t chEvtWaitAnyTimeout (eventmask_t mask, sysinterval_t timeout)`
Waits for any of the specified events.

8.2.1 Detailed Description

Nil RTOS main header file.

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

8.3 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 [chBSemResetI](#) ([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 [chBSemSignalI](#) ([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.

8.3.1 Detailed Description

Binary semaphores structures and macros.

8.4 chconf.h File Reference

Configuration file template.

Macros

Kernel parameters and options

- #define [CH_CFG_NUM_THREADS](#) 3
Number of user threads in the application.

System timer settings

- #define [CH_CFG_ST_RESOLUTION](#) 32
System time counter resolution.
- #define [CH_CFG_ST_FREQUENCY](#) 1000
System tick frequency.
- #define [CH_CFG_ST_TIMEDELTA](#) 0
Time delta constant for the tick-less mode.

Subsystem options

- #define `CH_CFG_USE_SEMAPHORES` TRUE
Semaphores APIs.
- #define `CH_CFG_USE_MUTEXES` FALSE
Mutexes APIs.
- #define `CH_CFG_USE_EVENTS` TRUE
Events Flags APIs.
- #define `CH_CFG_USE_MAILBOXES` TRUE
Mailboxes APIs.
- #define `CH_CFG_USE_MEMCORE` TRUE
Core Memory Manager APIs.
- #define `CH_CFG_USE_HEAP` TRUE
Heap Allocator APIs.
- #define `CH_CFG_USE_MEMPOOLS` TRUE
Memory Pools Allocator APIs.
- #define `CH_CFG_USE_OBJ_FIFOS` TRUE
Objects FIFOs APIs.
- #define `CH_CFG_MEMCORE_SIZE` 0
Managed RAM size.

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.

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
System assertions.
- #define `CH_DBG_ENABLE_STACK_CHECK` TRUE
Stack check.

Kernel hooks

- #define `CH_CFG_SYSTEM_INIT_HOOK()`
System initialization hook.
- #define `CH_CFG_THREAD_EXT_FIELDS` /* Add threads custom fields here.*/
Threads descriptor structure extension.
- #define `CH_CFG_THREAD_EXT_INIT_HOOK(tr)`
Threads initialization hook.

- `#define CH_CFG_IDLE_ENTER_HOOK()`
Idle thread enter hook.
- `#define CH_CFG_IDLE_LEAVE_HOOK()`
Idle thread leave hook.
- `#define CH_CFG_SYSTEM_HALT_HOOK(reason)`
System halt hook.

8.4.1 Detailed Description

Configuration file template.

A copy of this file must be placed in each project directory, it contains the application specific kernel settings.

8.5 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.

Variables

- `objects_factory_t ch_factory`
Factory object static instance.

8.5.1 Detailed Description

ChibiOS objects factory and registry code.

8.6 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_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
- Enables factory for objects FIFOs.*
 • #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.*

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_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.*
- 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.*

8.6.1 Detailed Description

ChibiOS objects factory structures and macros.

8.7 chfifo.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 `chFifoObjectInit` (`objects_fifo_t *ofp`, `size_t objsize`, `size_t objn`, `unsigned objalign`, `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 `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 `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.

8.7.1 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.

8.8 chheap.c File Reference

Heaps code.

```
#include "ch.h"
```

Functions

- void `_heap_init` (void)
Initializes the default heap.
- void `chHeapObjectInit` (`memory_heap_t` *heapp, void *buf, size_t size)
Initializes a memory heap from a static memory area.
- void * `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.

8.8.1 Detailed Description

Heaps code.

8.9 chheap.h File Reference

Heaps macros and structures.

Data Structures

- union `heap_header`
Memory heap block header.
- struct `memory_heap`
Structure describing a memory heap.

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.

8.9.1 Detailed Description

Heaps macros and structures.

8.10 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 `chMBPostI` (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.

8.10.1 Detailed Description

Mailboxes code.

8.11 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 `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 chMBPostI (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 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.

8.11.1 Detailed Description

Mailboxes macros and structures.

8.12 chmemcore.c File Reference

Core memory manager code.

```
#include "ch.h"
```

Functions

- `void _core_init (void)`
Low level memory manager initialization.
- `void * chCoreAllocAlignedWithOffsetl (size_t size, unsigned align, size_t offset)`
Allocates a memory block.
- `void * chCoreAllocAlignedWithOffset (size_t size, unsigned align, size_t offset)`
Allocates a memory block.
- `size_t chCoreGetStatusX (void)`
Core memory status.

Variables

- `memcore_t ch_memcore`
Memory core descriptor.

8.12.1 Detailed Description

Core memory manager code.

8.13 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.

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 * [chCoreAllocAlignedWithOffsetI](#) (size_t size, unsigned align, size_t offset)
Allocates a memory block.
- void * [chCoreAllocAlignedWithOffset](#) (size_t size, unsigned align, size_t offset)
Allocates a memory block.
- 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.

8.13.1 Detailed Description

Core memory manager macros and structures.

8.14 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.
- void [chGuardedPoolFree](#) ([guarded_memory_pool_t](#) *gmp, void *objp)
Releases an object into a guarded memory pool.

8.14.1 Detailed Description

Memory Pools code.

8.15 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)`
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.
- `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 chPoolAddI (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 void chGuardedPoolAdd (guarded_memory_pool_t *gmp, void *objp)`
Adds an object to a guarded memory pool.
- `static void chGuardedPoolAddI (guarded_memory_pool_t *gmp, void *objp)`
Adds an object to a guarded memory pool.
- `static void * chGuardedPoolAlloc (guarded_memory_pool_t *gmp)`
Allocates an object from a guarded memory pool.

8.15.1 Detailed Description

Memory Pools macros and structures.

Index

- [_BSEMAPHORE_DATA](#)
 - [Binary_semaphores, 128](#)
- [_CHIBIOS_NIL_](#)
 - [API, 26](#)
- [_GUARDEDMEMORYPOOL_DATA](#)
 - [Pools, 113](#)
- [_MAILBOX_DATA](#)
 - [Mailboxes, 90](#)
- [_MEMORYPOOL_DATA](#)
 - [Pools, 112](#)
- [_SEMAPHORE_DATA](#)
 - [API, 36](#)
- [_THREADS_QUEUE_DATA](#)
 - [API, 36](#)
- [_THREADS_QUEUE_DECL](#)
 - [API, 36](#)
- [__CH_STRINGIFY](#)
 - [API, 28](#)
- [_core_init](#)
 - [Memcore, 105](#)
- [_dbg_check_disable](#)
 - [API, 48](#)
- [_dbg_check_enable](#)
 - [API, 49](#)
- [_dbg_check_enter_isr](#)
 - [API, 51](#)
- [_dbg_check_leave_isr](#)
 - [API, 51](#)
- [_dbg_check_lock](#)
 - [API, 49](#)
- [_dbg_check_lock_from_isr](#)
 - [API, 50](#)
- [_dbg_check_suspend](#)
 - [API, 48](#)
- [_dbg_check_unlock](#)
 - [API, 50](#)
- [_dbg_check_unlock_from_isr](#)
 - [API, 50](#)
- [_factory_init](#)
 - [Objects_factory, 73](#)
- [_heap_init](#)
 - [Heaps, 85](#)
- [ALL_EVENTS](#)
 - [API, 27](#)
- [API, 19](#)
 - [_CHIBIOS_NIL_, 26](#)
 - [_SEMAPHORE_DATA, 36](#)
 - [_THREADS_QUEUE_DATA, 36](#)
 - [_THREADS_QUEUE_DECL, 36](#)
- [__CH_STRINGIFY, 28](#)
- [_dbg_check_disable, 48](#)
- [_dbg_check_enable, 49](#)
- [_dbg_check_enter_isr, 51](#)
- [_dbg_check_leave_isr, 51](#)
- [_dbg_check_lock, 49](#)
- [_dbg_check_lock_from_isr, 50](#)
- [_dbg_check_suspend, 48](#)
- [_dbg_check_unlock, 50](#)
- [_dbg_check_unlock_from_isr, 50](#)
- [ALL_EVENTS, 27](#)
- [CH_CFG_FACTORY_GENERIC_BUFFERS, 28](#)
- [CH_CFG_FACTORY_MAILBOXES, 28](#)
- [CH_CFG_FACTORY_MAX_NAMES_LENGTH, 28](#)
- [CH_CFG_FACTORY_OBJ_FIFOS, 28](#)
- [CH_CFG_FACTORY_OBJECTS_REGISTRY, 28](#)
- [CH_CFG_FACTORY_SEMAPHORES, 28](#)
- [CH_CFG_USE_FACTORY, 27](#)
- [CH_FAST_IRQ_HANDLER, 32](#)
- [CH_IRQ_EPILOGUE, 32](#)
- [CH_IRQ_HANDLER, 32](#)
- [CH_IRQ_IS_VALID_KERNEL_PRIORITY, 31](#)
- [CH_IRQ_IS_VALID_PRIORITY, 31](#)
- [CH_IRQ_PROLOGUE, 31](#)
- [CH_KERNEL_MAJOR, 26](#)
- [CH_KERNEL_MINOR, 26](#)
- [CH_KERNEL_PATCH, 26](#)
- [CH_KERNEL_STABLE, 26](#)
- [CH_KERNEL_VERSION, 26](#)
- [chDbgAssert, 46](#)
- [chDbgCheck, 46](#)
- [chDbgCheckClassI, 52](#)
- [chDbgCheckClassS, 52](#)
- [chEvtSignal, 66](#)
- [chEvtSignalI, 67](#)
- [chEvtWaitAnyTimeout, 68](#)
- [chSchDoReschedule, 57](#)
- [chSchGoSleepTimeoutS, 58](#)
- [chSchIsPreemptionRequired, 57](#)
- [chSchIsRescRequiredI, 39](#)
- [chSchReadyI, 56](#)
- [chSchRescheduleS, 57](#)
- [chSemFastSignalI, 43](#)
- [chSemFastWaitI, 43](#)
- [chSemGetCounterI, 44](#)
- [chSemObjectInit, 42](#)
- [chSemReset, 65](#)
- [chSemResetI, 66](#)
- [chSemSignal, 64](#)

- chSemSignal, 64
- chSemWait, 42
- chSemWaitTimeout, 62
- chSemWaitTimeoutS, 63
- chSemWaitS, 43
- chSysDisable, 37
- chSysEnable, 38
- chSysGetRealtimeCounterX, 37
- chSysGetStatusAndLockX, 54
- chSysHalt, 53
- chSysInit, 52
- chSysIsCounterWithinX, 55
- chSysLock, 38
- chSysLockFromISR, 38
- chSysPolledDelayX, 56
- chSysRestoreStatusX, 55
- chSysSuspend, 37
- chSysTimerHandlerI, 53
- chSysUnconditionalLock, 54
- chSysUnconditionalUnlock, 54
- chSysUnlock, 38
- chSysUnlockFromISR, 39
- chThdDequeueAll, 62
- chThdDequeueNextI, 61
- chThdDoDequeueNextI, 61
- chThdEnqueueTimeoutS, 60
- chThdGetSelfX, 40
- chThdQueueIsEmptyI, 42
- chThdQueueObjectInit, 41
- chThdResumeI, 59
- chThdSleep, 60
- chThdSleepMicroseconds, 40
- chThdSleepMilliseconds, 40
- chThdSleepSeconds, 40
- chThdSleepUntil, 60
- chThdSleepUntilS, 41
- chThdSleepS, 41
- chThdSuspendTimeoutS, 58
- chTimeAddX, 45
- chTimeDiffX, 45
- chTimeIsInRangeX, 45
- chVTGetSystemTimeX, 44
- chVTimeElapsedSinceX, 44
- EVENT_MASK, 27
- MEM_ALIGN_MASK, 29
- MEM_ALIGN_NEXT, 29
- MEM_ALIGN_PREV, 29
- MEM_IS_ALIGNED, 29
- MEM_IS_VALID_ALIGNMENT, 30
- MSG_OK, 26
- MSG_RESET, 26
- MSG_TIMEOUT, 26
- NIL_STATE_READY, 27
- NIL_STATE_SLEEPING, 27
- NIL_STATE_SUSP, 27
- NIL_STATE_WTOREVT, 27
- NIL_STATE_WTQUEUE, 27
- nil, 68
- nil_system_t, 48
- SEMAPHORE_DECL, 36
- semaphore_t, 48
- sysinterval_t, 47
- systime_t, 47
- THD_FUNCTION, 31
- THD_IDLE_BASE, 28
- THD_TABLE_BEGIN, 28
- THD_TABLE_ENTRY, 28
- THD_TABLE_END, 29
- THD_WORKING_AREA_SIZE, 30
- THD_WORKING_AREA, 30
- TIME_I2MS, 35
- TIME_I2US, 35
- TIME_I2S, 34
- TIME_IMMEDIATE, 26
- TIME_INFINITE, 27
- TIME_MAX_INTERVAL, 27
- TIME_MAX_SYSTIME, 27
- TIME_MS2I, 33
- TIME_S2I, 32
- TIME_US2I, 33
- tfunc_t, 48
- thread_config_t, 48
- thread_reference_t, 48
- thread_t, 47
- threads_queue_t, 47
- time_conv_t, 47
- align
 - memory_pool_t, 164
- arg
 - nil_thread_cfg, 170
- BSEMAPHORE_DECL
 - Binary_semaphores, 128
- binary_semaphore_t
 - Binary_semaphores, 128
- Binary_semaphores, 127
 - _BSEMAPHORE_DATA, 128
 - BSEMAPHORE_DECL, 128
 - binary_semaphore_t, 128
 - chBSemGetStatel, 134
 - chBSemObjectInit, 128
 - chBSemReset, 132
 - chBSemResetI, 131
 - chBSemSignal, 133
 - chBSemSignalI, 133
 - chBSemWait, 129
 - chBSemWaitTimeout, 130
 - chBSemWaitTimeoutS, 130
 - chBSemWaitS, 129
- buf_list
 - ch_objects_factory, 154
- buffer
 - ch_dyn_object, 150
 - mailbox_t, 160
- CH_CFG_FACTORY_GENERIC_BUFFERS
 - API, 28

- Configuration, [16](#)
- Objects_factory, [71](#)
- CH_CFG_FACTORY_MAILBOXES
 - API, [28](#)
 - Configuration, [16](#)
 - Objects_factory, [72](#)
- CH_CFG_FACTORY_MAX_NAMES_LENGTH
 - API, [28](#)
 - Configuration, [16](#)
 - Objects_factory, [71](#)
- CH_CFG_FACTORY_OBJ_FIFOS
 - API, [28](#)
 - Configuration, [16](#)
 - Objects_factory, [72](#)
- CH_CFG_FACTORY_OBJECTS_REGISTRY
 - API, [28](#)
 - Configuration, [16](#)
 - Objects_factory, [71](#)
- CH_CFG_FACTORY_SEMAPHORES
 - API, [28](#)
 - Configuration, [16](#)
 - Objects_factory, [72](#)
- CH_CFG_IDLE_ENTER_HOOK
 - Configuration, [17](#)
- CH_CFG_IDLE_LEAVE_HOOK
 - Configuration, [18](#)
- CH_CFG_MEMCORE_SIZE
 - Configuration, [15](#)
 - Memcore, [105](#)
- CH_CFG_NUM_THREADS
 - Configuration, [13](#)
- CH_CFG_ST_FREQUENCY
 - Configuration, [13](#)
- CH_CFG_ST_RESOLUTION
 - Configuration, [13](#)
- CH_CFG_ST_TIMEDELTA
 - Configuration, [14](#)
- CH_CFG_SYSTEM_HALT_HOOK
 - Configuration, [18](#)
- CH_CFG_SYSTEM_INIT_HOOK
 - Configuration, [17](#)
- CH_CFG_THREAD_EXT_FIELDS
 - Configuration, [17](#)
- CH_CFG_THREAD_EXT_INIT_HOOK
 - Configuration, [17](#)
- CH_CFG_USE_EVENTS
 - Configuration, [14](#)
- CH_CFG_USE_FACTORY
 - API, [27](#)
 - Configuration, [15](#)
- CH_CFG_USE_HEAP
 - Configuration, [15](#)
- CH_CFG_USE_MAILBOXES
 - Configuration, [14](#)
- CH_CFG_USE_MEMCORE
 - Configuration, [15](#)
- CH_CFG_USE_MEMPOOLS
 - Configuration, [15](#)
- CH_CFG_USE_MUTEXES
 - Configuration, [14](#)
- CH_CFG_USE_OBJ_FIFOS
 - Configuration, [15](#)
- CH_CFG_USE_SEMAPHORES
 - Configuration, [14](#)
- CH_DBG_ENABLE_ASSERTS
 - Configuration, [17](#)
- CH_DBG_ENABLE_CHECKS
 - Configuration, [16](#)
- CH_DBG_ENABLE_STACK_CHECK
 - Configuration, [17](#)
- CH_DBG_STATISTICS
 - Configuration, [16](#)
- CH_DBG_SYSTEM_STATE_CHECK
 - Configuration, [16](#)
- CH_FAST_IRQ_HANDLER
 - API, [32](#)
- CH_HEAP_ALIGNMENT
 - Heaps, [85](#)
- CH_HEAP_AREA
 - Heaps, [85](#)
- CH_IRQ_EPILOGUE
 - API, [32](#)
- CH_IRQ_HANDLER
 - API, [32](#)
- CH_IRQ_IS_VALID_KERNEL_PRIORITY
 - API, [31](#)
- CH_IRQ_IS_VALID_PRIORITY
 - API, [31](#)
- CH_IRQ_PROLOGUE
 - API, [31](#)
- CH_KERNEL_MAJOR
 - API, [26](#)
- CH_KERNEL_MINOR
 - API, [26](#)
- CH_KERNEL_PATCH
 - API, [26](#)
- CH_KERNEL_STABLE
 - API, [26](#)
- CH_KERNEL_VERSION
 - API, [26](#)
- ch.c, [173](#)
- ch.h, [175](#)
- ch_binary_semaphore, [145](#)
- ch_dyn_element, [146](#)
 - next, [147](#)
 - refs, [147](#)
- ch_dyn_list, [147](#)
- ch_dyn_mailbox, [148](#)
 - element, [149](#)
 - mbx, [149](#)
 - msgbuf, [149](#)
- ch_dyn_object, [149](#)
 - buffer, [150](#)
 - element, [150](#)
- ch_dyn_objects_fifo, [150](#)
 - element, [151](#)

- fifo, [151](#)
 - msgbuf, [152](#)
- ch_dyn_semaphore, [152](#)
 - element, [153](#)
 - sem, [153](#)
- ch_factory
 - Objects_factory, [83](#)
- ch_memcore
 - Memcore, [110](#)
- ch_objects_factory, [153](#)
 - buf_list, [154](#)
 - fifo_list, [154](#)
 - mbx_list, [154](#)
 - mtx, [154](#)
 - obj_list, [154](#)
 - obj_pool, [154](#)
 - sem_list, [154](#)
 - sem_pool, [154](#)
- ch_objects_fifo, [155](#)
 - free, [156](#)
 - mbx, [156](#)
- ch_registered_static_object, [156](#)
 - element, [157](#)
 - objp, [157](#)
- chBSemGetStatel
 - Binary_semaphores, [134](#)
- chBSemObjectInit
 - Binary_semaphores, [128](#)
- chBSemReset
 - Binary_semaphores, [132](#)
- chBSemResetl
 - Binary_semaphores, [131](#)
- chBSemSignal
 - Binary_semaphores, [133](#)
- chBSemSignalI
 - Binary_semaphores, [133](#)
- chBSemWait
 - Binary_semaphores, [129](#)
- chBSemWaitTimeout
 - Binary_semaphores, [130](#)
- chBSemWaitTimeoutS
 - Binary_semaphores, [130](#)
- chBSemWaitS
 - Binary_semaphores, [129](#)
- chCoreAlloc
 - Memcore, [109](#)
- chCoreAllocAligned
 - Memcore, [108](#)
- chCoreAllocAlignedWithOffset
 - Memcore, [106](#)
- chCoreAllocAlignedWithOffsetI
 - Memcore, [105](#)
- chCoreAllocAlignedI
 - Memcore, [107](#)
- chCoreAllocI
 - Memcore, [108](#)
- chCoreGetStatusX
 - Memcore, [107](#)
- chDbgAssert
 - API, [46](#)
- chDbgCheck
 - API, [46](#)
- chDbgCheckClassI
 - API, [52](#)
- chDbgCheckClassS
 - API, [52](#)
- chEvtSignal
 - API, [66](#)
- chEvtSignalI
 - API, [67](#)
- chEvtWaitAnyTimeout
 - API, [68](#)
- chFactoryCreateBuffer
 - Objects_factory, [75](#)
- chFactoryCreateMailbox
 - Objects_factory, [78](#)
- chFactoryCreateObjectsFIFO
 - Objects_factory, [79](#)
- chFactoryCreateSemaphore
 - Objects_factory, [76](#)
- chFactoryDuplicateReference
 - Objects_factory, [81](#)
- chFactoryFindBuffer
 - Objects_factory, [76](#)
- chFactoryFindMailbox
 - Objects_factory, [78](#)
- chFactoryFindObject
 - Objects_factory, [74](#)
- chFactoryFindObjectByPointer
 - Objects_factory, [74](#)
- chFactoryFindObjectsFIFO
 - Objects_factory, [80](#)
- chFactoryFindSemaphore
 - Objects_factory, [77](#)
- chFactoryGetBuffer
 - Objects_factory, [82](#)
- chFactoryGetBufferSize
 - Objects_factory, [81](#)
- chFactoryGetMailbox
 - Objects_factory, [83](#)
- chFactoryGetObject
 - Objects_factory, [81](#)
- chFactoryGetObjectsFIFO
 - Objects_factory, [83](#)
- chFactoryGetSemaphore
 - Objects_factory, [82](#)
- chFactoryRegisterObject
 - Objects_factory, [73](#)
- chFactoryReleaseBuffer
 - Objects_factory, [76](#)
- chFactoryReleaseMailbox
 - Objects_factory, [79](#)
- chFactoryReleaseObject
 - Objects_factory, [75](#)
- chFactoryReleaseObjectsFIFO
 - Objects_factory, [80](#)

- chFactoryReleaseSemaphore
 - Objects_factory, [77](#)
- chFifoObjectInit
 - Objects_fifo, [136](#)
- chFifoReceiveObjectTimeout
 - Objects_fifo, [143](#)
- chFifoReceiveObjectTimeoutS
 - Objects_fifo, [142](#)
- chFifoReceiveObjectI
 - Objects_fifo, [141](#)
- chFifoReturnObject
 - Objects_fifo, [139](#)
- chFifoReturnObjectI
 - Objects_fifo, [138](#)
- chFifoSendObject
 - Objects_fifo, [141](#)
- chFifoSendObjectI
 - Objects_fifo, [139](#)
- chFifoSendObjectS
 - Objects_fifo, [140](#)
- chFifoTakeObjectTimeout
 - Objects_fifo, [138](#)
- chFifoTakeObjectTimeoutS
 - Objects_fifo, [137](#)
- chFifoTakeObjectI
 - Objects_fifo, [136](#)
- chGuardedPoolAdd
 - Pools, [124](#)
- chGuardedPoolAddI
 - Pools, [124](#)
- chGuardedPoolAllocTimeout
 - Pools, [119](#)
- chGuardedPoolAllocTimeoutS
 - Pools, [118](#)
- chGuardedPoolAllocI
 - Pools, [125](#)
- chGuardedPoolFree
 - Pools, [120](#)
- chGuardedPoolFreeI
 - Pools, [120](#)
- chGuardedPoolLoadArray
 - Pools, [117](#)
- chGuardedPoolObjectInit
 - Pools, [123](#)
- chGuardedPoolObjectInitAligned
 - Pools, [117](#)
- chHeapAlloc
 - Heaps, [87](#)
- chHeapAllocAligned
 - Heaps, [86](#)
- chHeapFree
 - Heaps, [87](#)
- chHeapGetSize
 - Heaps, [88](#)
- chHeapObjectInit
 - Heaps, [86](#)
- chHeapStatus
 - Heaps, [87](#)
- chMBFetchTimeout
 - Mailboxes, [98](#)
- chMBFetchTimeoutS
 - Mailboxes, [99](#)
- chMBFetchI
 - Mailboxes, [100](#)
- chMBGetFreeCountI
 - Mailboxes, [101](#)
- chMBGetSizeI
 - Mailboxes, [101](#)
- chMBGetUsedCountI
 - Mailboxes, [101](#)
- chMBOBJECTInit
 - Mailboxes, [91](#)
- chMBPeekI
 - Mailboxes, [102](#)
- chMBPostAheadTimeout
 - Mailboxes, [95](#)
- chMBPostAheadTimeoutS
 - Mailboxes, [96](#)
- chMBPostAheadI
 - Mailboxes, [97](#)
- chMBPostTimeout
 - Mailboxes, [92](#)
- chMBPostTimeoutS
 - Mailboxes, [93](#)
- chMBPostI
 - Mailboxes, [94](#)
- chMBReset
 - Mailboxes, [91](#)
- chMBResetI
 - Mailboxes, [92](#)
- chMBResumeX
 - Mailboxes, [102](#)
- chPoolAdd
 - Pools, [122](#)
- chPoolAddI
 - Pools, [122](#)
- chPoolAlloc
 - Pools, [115](#)
- chPoolAllocI
 - Pools, [114](#)
- chPoolFree
 - Pools, [116](#)
- chPoolFreeI
 - Pools, [116](#)
- chPoolLoadArray
 - Pools, [114](#)
- chPoolObjectInit
 - Pools, [121](#)
- chPoolObjectInitAligned
 - Pools, [113](#)
- chSchDoReschedule
 - API, [57](#)
- chSchGoSleepTimeoutS
 - API, [58](#)
- chSchIsPreemptionRequired
 - API, [57](#)

- chSchIsRescRequiredI
 - API, [39](#)
- chSchReadyI
 - API, [56](#)
- chSchRescheduleS
 - API, [57](#)
- chSemFastSignalI
 - API, [43](#)
- chSemFastWaitI
 - API, [43](#)
- chSemGetCounterI
 - API, [44](#)
- chSemObjectInit
 - API, [42](#)
- chSemReset
 - API, [65](#)
- chSemResetI
 - API, [66](#)
- chSemSignal
 - API, [64](#)
- chSemSignalI
 - API, [64](#)
- chSemWait
 - API, [42](#)
- chSemWaitTimeout
 - API, [62](#)
- chSemWaitTimeoutS
 - API, [63](#)
- chSemWaitS
 - API, [43](#)
- chSysDisable
 - API, [37](#)
- chSysEnable
 - API, [38](#)
- chSysGetRealtimeCounterX
 - API, [37](#)
- chSysGetStatusAndLockX
 - API, [54](#)
- chSysHalt
 - API, [53](#)
- chSysInit
 - API, [52](#)
- chSysIsCounterWithinX
 - API, [55](#)
- chSysLock
 - API, [38](#)
- chSysLockFromISR
 - API, [38](#)
- chSysPolledDelayX
 - API, [56](#)
- chSysRestoreStatusX
 - API, [55](#)
- chSysSuspend
 - API, [37](#)
- chSysTimerHandlerI
 - API, [53](#)
- chSysUnconditionalLock
 - API, [54](#)
- chSysUnconditionalUnlock
 - API, [54](#)
- chSysUnlock
 - API, [38](#)
- chSysUnlockFromISR
 - API, [39](#)
- chThdDequeueAllI
 - API, [62](#)
- chThdDequeueNextI
 - API, [61](#)
- chThdDoDequeueNextI
 - API, [61](#)
- chThdEnqueueTimeoutS
 - API, [60](#)
- chThdGetSelfX
 - API, [40](#)
- chThdQueueIsEmptyI
 - API, [42](#)
- chThdQueueObjectInit
 - API, [41](#)
- chThdResumeI
 - API, [59](#)
- chThdSleep
 - API, [60](#)
- chThdSleepMicroseconds
 - API, [40](#)
- chThdSleepMilliseconds
 - API, [40](#)
- chThdSleepSeconds
 - API, [40](#)
- chThdSleepUntil
 - API, [60](#)
- chThdSleepUntilS
 - API, [41](#)
- chThdSleepS
 - API, [41](#)
- chThdSuspendTimeoutS
 - API, [58](#)
- chTimeAddX
 - API, [45](#)
- chTimeDiffX
 - API, [45](#)
- chTimeIsInRangeX
 - API, [45](#)
- chVTGetSystemTimeX
 - API, [44](#)
- chVTimeElapsedSinceX
 - API, [44](#)
- chbsem.h, [181](#)
- chconf.h, [182](#)
- chfactory.c, [184](#)
- chfactory.h, [185](#)
- chfifo.h, [187](#)
- chheap.c, [188](#)
- chheap.h, [189](#)
- chmbboxes.c, [190](#)
- chmbboxes.h, [191](#)
- chmemcore.c, [192](#)

- chmemcore.h, 192
- chmempools.c, 193
- chmempools.h, 194
- cnt
 - mailbox_t, 160
 - nil_threads_queue, 171
- Configuration, 12
 - CH_CFG_FACTORY_GENERIC_BUFFERS, 16
 - CH_CFG_FACTORY_MAILBOXES, 16
 - CH_CFG_FACTORY_MAX_NAMES_LENGTH, 16
 - CH_CFG_FACTORY_OBJ_FIFOS, 16
 - CH_CFG_FACTORY_OBJECTS_REGISTRY, 16
 - CH_CFG_FACTORY_SEMAPHORES, 16
 - CH_CFG_IDLE_ENTER_HOOK, 17
 - CH_CFG_IDLE_LEAVE_HOOK, 18
 - CH_CFG_MEMCORE_SIZE, 15
 - CH_CFG_NUM_THREADS, 13
 - CH_CFG_ST_FREQUENCY, 13
 - CH_CFG_ST_RESOLUTION, 13
 - CH_CFG_ST_TIMEDELTA, 14
 - CH_CFG_SYSTEM_HALT_HOOK, 18
 - CH_CFG_SYSTEM_INIT_HOOK, 17
 - CH_CFG_THREAD_EXT_FIELDS, 17
 - CH_CFG_THREAD_EXT_INIT_HOOK, 17
 - CH_CFG_USE_EVENTS, 14
 - CH_CFG_USE_FACTORY, 15
 - CH_CFG_USE_HEAP, 15
 - CH_CFG_USE_MAILBOXES, 14
 - CH_CFG_USE_MEMCORE, 15
 - CH_CFG_USE_MEMPOOLS, 15
 - CH_CFG_USE_Mutexes, 14
 - CH_CFG_USE_OBJ_FIFOS, 15
 - CH_CFG_USE_SEMAPHORES, 14
 - CH_DBG_ENABLE_ASSERTS, 17
 - CH_DBG_ENABLE_CHECKS, 16
 - CH_DBG_ENABLE_STACK_CHECK, 17
 - CH_DBG_STATISTICS, 16
 - CH_DBG_SYSTEM_STATE_CHECK, 16
- ctx
 - nil_thread, 168
- current
 - nil_system, 166
- dbg_panic_msg
 - nil_system, 166
- default_heap
 - Heaps, 88
- dyn_buffer_t
 - Objects_factory, 72
- dyn_element_t
 - Objects_factory, 72
- dyn_list_t
 - Objects_factory, 72
- dyn_mailbox_t
 - Objects_factory, 72
- dyn_objects_fifo_t
 - Objects_factory, 73
- dyn_semaphore_t
 - Objects_factory, 72
- EVENT_MASK
 - API, 27
- element
 - ch_dyn_mailbox, 149
 - ch_dyn_object, 150
 - ch_dyn_objects_fifo, 151
 - ch_dyn_semaphore, 153
 - ch_registered_static_object, 157
- endmem
 - memcore_t, 161
- epmask
 - nil_thread, 169
- ewmask
 - nil_thread, 169
- fifo
 - ch_dyn_objects_fifo, 151
- fifo_list
 - ch_objects_factory, 154
- free
 - ch_objects_fifo, 156
- funcp
 - nil_thread_cfg, 170
- GUARDEDMEMORYPOOL_DECL
 - Pools, 113
- guarded_memory_pool_t, 157
 - pool, 158
 - sem, 158
- header
 - memory_heap, 162
- heap
 - heap_header, 159
- heap_header, 158
 - heap, 159
 - next, 159
 - pages, 159
 - size, 159
- heap_header_t
 - Heaps, 85
- Heaps, 84
 - _heap_init, 85
 - CH_HEAP_ALIGNMENT, 85
 - CH_HEAP_AREA, 85
 - chHeapAlloc, 87
 - chHeapAllocAligned, 86
 - chHeapFree, 87
 - chHeapGetSize, 88
 - chHeapObjectInit, 86
 - chHeapStatus, 87
 - default_heap, 88
 - heap_header_t, 85
 - memory_heap_t, 85
- isr_cnt
 - nil_system, 166
- lasttime

- nil_system, 166
- lock_cnt
 - nil_system, 166
- MAILBOX_DECL
 - Mailboxes, 90
- MEM_ALIGN_MASK
 - API, 29
- MEM_ALIGN_NEXT
 - API, 29
- MEM_ALIGN_PREV
 - API, 29
- MEM_IS_ALIGNED
 - API, 29
- MEM_IS_VALID_ALIGNMENT
 - API, 30
- MEMORYPOOL_DECL
 - Pools, 112
- MSG_OK
 - API, 26
- MSG_RESET
 - API, 26
- MSG_TIMEOUT
 - API, 26
- mailbox_t, 159
 - buffer, 160
 - cnt, 160
 - qr, 161
 - qw, 160
 - rdptr, 160
 - reset, 160
 - top, 160
 - wrptr, 160
- Mailboxes, 89
 - _MAILBOX_DATA, 90
 - chMBFetchTimeout, 98
 - chMBFetchTimeoutS, 99
 - chMBFetchI, 100
 - chMBGetFreeCountI, 101
 - chMBGetSizeI, 101
 - chMBGetUsedCountI, 101
 - chMBOBJECTInit, 91
 - chMBPeekI, 102
 - chMBPostAheadTimeout, 95
 - chMBPostAheadTimeoutS, 96
 - chMBPostAheadI, 97
 - chMBPostTimeout, 92
 - chMBPostTimeoutS, 93
 - chMBPostI, 94
 - chMBReset, 91
 - chMBResetI, 92
 - chMBResumeX, 102
 - MAILBOX_DECL, 90
- mbx
 - ch_dyn_mailbox, 149
 - ch_objects_fifo, 156
- mbx_list
 - ch_objects_factory, 154
- Memcore, 104
 - _core_init, 105
 - CH_CFG_MEMCORE_SIZE, 105
 - ch_memcore, 110
 - chCoreAlloc, 109
 - chCoreAllocAligned, 108
 - chCoreAllocAlignedWithOffset, 106
 - chCoreAllocAlignedWithOffsetI, 105
 - chCoreAllocAlignedI, 107
 - chCoreAllocI, 108
 - chCoreGetStatusX, 107
 - memgetfunc2_t, 105
 - memgetfunc_t, 105
- memcore_t, 161
 - endmem, 161
 - nextmem, 161
- memgetfunc2_t
 - Memcore, 105
- memgetfunc_t
 - Memcore, 105
- memory_heap, 162
 - header, 162
 - mtx, 162
 - provider, 162
- memory_heap_t
 - Heaps, 85
- memory_pool_t, 163
 - align, 164
 - next, 163
 - object_size, 163
 - provider, 164
- msg
 - nil_thread, 168
- msgbuf
 - ch_dyn_mailbox, 149
 - ch_dyn_objects_fifo, 152
- mtx
 - ch_objects_factory, 154
 - memory_heap, 162
- NIL Kernel, 11
- NIL_STATE_READY
 - API, 27
- NIL_STATE_SLEEPING
 - API, 27
- NIL_STATE_SUSP
 - API, 27
- NIL_STATE_WTOREVT
 - API, 27
- NIL_STATE_WTQUEUE
 - API, 27
- namep
 - nil_thread_cfg, 170
- next
 - ch_dyn_element, 147
 - heap_header, 159
 - memory_pool_t, 163
 - nil_system, 166
 - pool_header, 172
- nextmem

- memcore_t, 161
- nexttime
 - nil_system, 166
- nil
 - API, 68
- nil_system, 164
 - current, 166
 - dbg_panic_msg, 166
 - isr_cnt, 166
 - lasttime, 166
 - lock_cnt, 166
 - next, 166
 - nexttime, 166
 - systime, 166
 - threads, 167
- nil_system_t
 - API, 48
- nil_thread, 167
 - ctx, 168
 - epmask, 169
 - ewmask, 169
 - msg, 168
 - p, 168
 - semp, 168
 - state, 168
 - timeout, 169
 - tqp, 168
 - trp, 168
 - wabase, 169
- nil_thread_cfg, 169
 - arg, 170
 - funcp, 170
 - namep, 170
 - wbase, 170
 - wend, 170
- nil_threads_queue, 170
 - cnt, 171
- obj_list
 - ch_objects_factory, 154
- obj_pool
 - ch_objects_factory, 154
- object_size
 - memory_pool_t, 163
- Objects_factory, 69
 - _factory_init, 73
 - CH_CFG_FACTORY_GENERIC_BUFFERS, 71
 - CH_CFG_FACTORY_MAILBOXES, 72
 - CH_CFG_FACTORY_MAX_NAMES_LENGTH, 71
 - CH_CFG_FACTORY_OBJ_FIFOS, 72
 - CH_CFG_FACTORY_OBJECTS_REGISTRY, 71
 - CH_CFG_FACTORY_SEMAPHORES, 72
 - ch_factory, 83
 - chFactoryCreateBuffer, 75
 - chFactoryCreateMailbox, 78
 - chFactoryCreateObjectsFIFO, 79
 - chFactoryCreateSemaphore, 76
 - chFactoryDuplicateReference, 81
 - chFactoryFindBuffer, 76
 - chFactoryFindMailbox, 78
 - chFactoryFindObject, 74
 - chFactoryFindObjectByPointer, 74
 - chFactoryFindObjectsFIFO, 80
 - chFactoryFindSemaphore, 77
 - chFactoryGetBuffer, 82
 - chFactoryGetBufferSize, 81
 - chFactoryGetMailbox, 83
 - chFactoryGetObject, 81
 - chFactoryGetObjectsFIFO, 83
 - chFactoryGetSemaphore, 82
 - chFactoryRegisterObject, 73
 - chFactoryReleaseBuffer, 76
 - chFactoryReleaseMailbox, 79
 - chFactoryReleaseObject, 75
 - chFactoryReleaseObjectsFIFO, 80
 - chFactoryReleaseSemaphore, 77
 - dyn_buffer_t, 72
 - dyn_element_t, 72
 - dyn_list_t, 72
 - dyn_mailbox_t, 72
 - dyn_objects_fifo_t, 73
 - dyn_semaphore_t, 72
 - objects_factory_t, 73
 - registered_object_t, 72
- objects_factory_t
 - Objects_factory, 73
- Objects_fifo, 135
 - chFifoObjectInit, 136
 - chFifoReceiveObjectTimeout, 143
 - chFifoReceiveObjectTimeoutS, 142
 - chFifoReceiveObjectI, 141
 - chFifoReturnObject, 139
 - chFifoReturnObjectI, 138
 - chFifoSendObject, 141
 - chFifoSendObjectI, 139
 - chFifoSendObjectS, 140
 - chFifoTakeObjectTimeout, 138
 - chFifoTakeObjectTimeoutS, 137
 - chFifoTakeObjectI, 136
 - objects_fifo_t, 135
- objects_fifo_t
 - Objects_fifo, 135
- objp
 - ch_registered_static_object, 157
- p
 - nil_thread, 168
- pages
 - heap_header, 159
- pool
 - guarded_memory_pool_t, 158
- pool_header, 172
 - next, 172
- Pools, 111
 - _GUARDEDMEMORYPOOL_DATA, 113
 - _MEMORYPOOL_DATA, 112
 - chGuardedPoolAdd, 124
 - chGuardedPoolAddI, 124

- chGuardedPoolAllocTimeout, 119
- chGuardedPoolAllocTimeoutS, 118
- chGuardedPoolAlloc, 125
- chGuardedPoolFree, 120
- chGuardedPoolFreeI, 120
- chGuardedPoolLoadArray, 117
- chGuardedPoolObjectInit, 123
- chGuardedPoolObjectInitAligned, 117
- chPoolAdd, 122
- chPoolAddI, 122
- chPoolAlloc, 115
- chPoolAllocI, 114
- chPoolFree, 116
- chPoolFreeI, 116
- chPoolLoadArray, 114
- chPoolObjectInit, 121
- chPoolObjectInitAligned, 113
- GUARDEDMEMORYPOOL_DECL, 113
- MEMORYPOOL_DECL, 112
- provider
 - memory_heap, 162
 - memory_pool_t, 164
- qr
 - mailbox_t, 161
- qw
 - mailbox_t, 160
- rdptr
 - mailbox_t, 160
- refs
 - ch_dyn_semaphore, 147
- registered_object_t
 - Objects_factory, 72
- reset
 - mailbox_t, 160
- SEMAPHORE_DECL
 - API, 36
- sem
 - ch_dyn_semaphore, 153
 - guarded_memory_pool_t, 158
- sem_list
 - ch_objects_factory, 154
- sem_pool
 - ch_objects_factory, 154
- semaphore_t
 - API, 48
- semp
 - nil_thread, 168
- size
 - heap_header, 159
- state
 - nil_thread, 168
- sysinterval_t
 - API, 47
- sysptime
 - nil_system, 166
- sysptime_t
 - API, 47
- THD_FUNCTION
 - API, 31
- THD_IDLE_BASE
 - API, 28
- THD_TABLE_BEGIN
 - API, 28
- THD_TABLE_ENTRY
 - API, 28
- THD_TABLE_END
 - API, 29
- THD_WORKING_AREA_SIZE
 - API, 30
- THD_WORKING_AREA
 - API, 30
- TIME_I2MS
 - API, 35
- TIME_I2US
 - API, 35
- TIME_I2S
 - API, 34
- TIME_IMMEDIATE
 - API, 26
- TIME_INFINITE
 - API, 27
- TIME_MAX_INTERVAL
 - API, 27
- TIME_MAX_SYSTIME
 - API, 27
- TIME_MS2I
 - API, 33
- TIME_S2I
 - API, 32
- TIME_US2I
 - API, 33
- tfunc_t
 - API, 48
- thread_config_t
 - API, 48
- thread_reference_t
 - API, 48
- thread_t
 - API, 47
- threads
 - nil_system, 167
- threads_queue_t
 - API, 47
- time_conv_t
 - API, 47
- timeout
 - nil_thread, 169
- top
 - mailbox_t, 160
- tpq
 - nil_thread, 168
- trp
 - nil_thread, 168

wabase
 nil_thread, [169](#)
wbase
 nil_thread_cfg, [170](#)
wend
 nil_thread_cfg, [170](#)
wrptr
 mailbox_t, [160](#)