

ChibiOS/NIL

3.2.2

Reference Manual

Thu Apr 30 2020 11:07:20

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	14
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	15
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_USE_PIPES	15
6.2.2.14	CH_CFG_MEMCORE_SIZE	16
6.2.2.15	CH_CFG_USE_FACTORY	16
6.2.2.16	CH_CFG_FACTORY_MAX_NAMES_LENGTH	16
6.2.2.17	CH_CFG_FACTORY_OBJECTS_REGISTRY	16
6.2.2.18	CH_CFG_FACTORY_GENERIC_BUFFERS	16
6.2.2.19	CH_CFG_FACTORY_SEMAPHORES	16
6.2.2.20	CH_CFG_FACTORY_MAILBOXES	16
6.2.2.21	CH_CFG_FACTORY_OBJ_FIFOS	16
6.2.2.22	CH_CFG_FACTORY_PIPES	16
6.2.2.23	CH_DBG_STATISTICS	17
6.2.2.24	CH_DBG_SYSTEM_STATE_CHECK	17
6.2.2.25	CH_DBG_ENABLE_CHECKS	17
6.2.2.26	CH_DBG_ENABLE_ASSERTS	17
6.2.2.27	CH_DBG_ENABLE_STACK_CHECK	17
6.2.2.28	CH_CFG_SYSTEM_INIT_HOOK	17
6.2.2.29	CH_CFG_THREAD_EXT_FIELDS	17
6.2.2.30	CH_CFG_THREAD_EXT_INIT_HOOK	18
6.2.2.31	CH_CFG_IDLE_ENTER_HOOK	18
6.2.2.32	CH_CFG_IDLE_LEAVE_HOOK	18
6.2.2.33	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	FALSE	26
6.3.2.8	TRUE	27
6.3.2.9	MSG_OK	27
6.3.2.10	MSG_TIMEOUT	27
6.3.2.11	MSG_RESET	27
6.3.2.12	TIME_IMMEDIATE	27
6.3.2.13	TIME_INFINITE	27
6.3.2.14	TIME_MAX_INTERVAL	27

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

6.3.2.55	<code>_THREADS_QUEUE_DATA</code>	36
6.3.2.56	<code>_THREADS_QUEUE_DECL</code>	36
6.3.2.57	<code>_SEMAPHORE_DATA</code>	36
6.3.2.58	<code>SEMAPHORE_DECL</code>	37
6.3.2.59	<code>chSysGetRealtimeCounterX</code>	37
6.3.2.60	<code>chSysDisable</code>	37
6.3.2.61	<code>chSysSuspend</code>	37
6.3.2.62	<code>chSysEnable</code>	38
6.3.2.63	<code>chSysLock</code>	38
6.3.2.64	<code>chSysUnlock</code>	38
6.3.2.65	<code>chSysLockFromISR</code>	39
6.3.2.66	<code>chSysUnlockFromISR</code>	39
6.3.2.67	<code>chSchIsRescRequiredI</code>	39
6.3.2.68	<code>chThdGetSelfX</code>	40
6.3.2.69	<code>chThdSleepSeconds</code>	40
6.3.2.70	<code>chThdSleepMilliseconds</code>	40
6.3.2.71	<code>chThdSleepMicroseconds</code>	40
6.3.2.72	<code>chThdSleepS</code>	41
6.3.2.73	<code>chThdSleepUntilS</code>	41
6.3.2.74	<code>chThdQueueObjectInit</code>	41
6.3.2.75	<code>chThdQueueIsEmptyI</code>	42
6.3.2.76	<code>chSemObjectInit</code>	42
6.3.2.77	<code>chSemWait</code>	42
6.3.2.78	<code>chSemWaitS</code>	43
6.3.2.79	<code>chSemFastWaitI</code>	43
6.3.2.80	<code>chSemFastSignalI</code>	44
6.3.2.81	<code>chSemGetCounterI</code>	44
6.3.2.82	<code>chVTGetSystemTimeX</code>	44
6.3.2.83	<code>chVTTimeElapsedSinceX</code>	44
6.3.2.84	<code>chTimeAddX</code>	45
6.3.2.85	<code>chTimeDiffX</code>	45
6.3.2.86	<code>chTimeIsInRangeX</code>	45
6.3.2.87	<code>chDbgCheck</code>	46
6.3.2.88	<code>chDbgAssert</code>	46
6.3.3	Typedef Documentation	47
6.3.3.1	<code>system_t</code>	47
6.3.3.2	<code>sysinterval_t</code>	47
6.3.3.3	<code>time_conv_t</code>	47
6.3.3.4	<code>thread_t</code>	47
6.3.3.5	<code>threads_queue_t</code>	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	nil_system_t	48
6.3.4	Function Documentation	48
6.3.4.1	nil_find_thread(tstate_t state, void *p)	48
6.3.4.2	nil_ready_all(void *p, cnt_t cnt, msg_t msg)	48
6.3.4.3	_dbg_check_disable(void)	49
6.3.4.4	_dbg_check_suspend(void)	49
6.3.4.5	_dbg_check_enable(void)	50
6.3.4.6	_dbg_check_lock(void)	50
6.3.4.7	_dbg_check_unlock(void)	51
6.3.4.8	_dbg_check_lock_from_isr(void)	51
6.3.4.9	_dbg_check_unlock_from_isr(void)	51
6.3.4.10	_dbg_check_enter_isr(void)	52
6.3.4.11	_dbg_check_leave_isr(void)	52
6.3.4.12	chDbgCheckClassI(void)	53
6.3.4.13	chDbgCheckClassS(void)	53
6.3.4.14	chSysInit(void)	53
6.3.4.15	chSysHalt(const char *reason)	54
6.3.4.16	chSysTimerHandlerI(void)	54
6.3.4.17	chSysUnconditionalLock(void)	55
6.3.4.18	chSysUnconditionalUnlock(void)	55
6.3.4.19	chSysGetStatusAndLockX(void)	55
6.3.4.20	chSysRestoreStatusX(syssts_t sts)	56
6.3.4.21	chSysIsCounterWithinX(rtcnt_t cnt, rtcnt_t start, rtcnt_t end)	56
6.3.4.22	chSysPolledDelayX(rtcnt_t cycles)	57
6.3.4.23	chSchReadyI(thread_t *tp, msg_t msg)	57
6.3.4.24	chSchIsPreemptionRequired(void)	58
6.3.4.25	chSchDoReschedule(void)	58
6.3.4.26	chSchRescheduleS(void)	58
6.3.4.27	chSchGoSleepTimeoutS(tstate_t newstate, sysinterval_t timeout)	59
6.3.4.28	chThdSuspendTimeoutS(thread_reference_t *trp, sysinterval_t timeout)	60
6.3.4.29	chThdResumeI(thread_reference_t *trp, msg_t msg)	60
6.3.4.30	chThdResume(thread_reference_t *trp, msg_t msg)	61
6.3.4.31	chThdSleep(sysinterval_t timeout)	61
6.3.4.32	chThdSleepUntil(systime_t abstime)	62
6.3.4.33	chThdEnqueueTimeoutS(threads_queue_t *tqp, sysinterval_t timeout)	62
6.3.4.34	chThdDoDequeueNextI(threads_queue_t *tqp, msg_t msg)	63

6.3.4.35	chThdDequeueNextl(threads_queue_t *tqp, msg_t msg)	63
6.3.4.36	chThdDequeueAll(threads_queue_t *tqp, msg_t msg)	64
6.3.4.37	chSemWaitTimeout(semaphore_t *sp, sysinterval_t timeout)	64
6.3.4.38	chSemWaitTimeoutS(semaphore_t *sp, sysinterval_t timeout)	65
6.3.4.39	chSemSignal(semaphore_t *sp)	66
6.3.4.40	chSemSignalI(semaphore_t *sp)	66
6.3.4.41	chSemReset(semaphore_t *sp, cnt_t n)	67
6.3.4.42	chSemResetI(semaphore_t *sp, cnt_t n)	68
6.3.4.43	chEvtSignal(thread_t *tp, eventmask_t mask)	68
6.3.4.44	chEvtSignalI(thread_t *tp, eventmask_t mask)	69
6.3.4.45	chEvtWaitAnyTimeout(eventmask_t mask, sysinterval_t timeout)	69
6.3.5	Variable Documentation	70
6.3.5.1	nil	70
6.4	OS Library	71
6.4.1	Detailed Description	71
6.5	Version Numbers and Identification	72
6.5.1	Detailed Description	72
6.5.2	Macro Definition Documentation	72
6.5.2.1	_CHIBIOS_OSLIB_	72
6.5.2.2	CH_OSLIB_STABLE	72
6.5.2.3	CH_OSLIB_VERSION	72
6.5.2.4	CH_OSLIB_MAJOR	72
6.5.2.5	CH_OSLIB_MINOR	72
6.5.2.6	CH_OSLIB_PATCH	72
6.6	Synchronization	73
6.6.1	Detailed Description	73
6.7	Binary Semaphores	74
6.7.1	Detailed Description	74
6.7.2	Macro Definition Documentation	74
6.7.2.1	_BSEMAPHORE_DATA	74
6.7.2.2	BSEMAPHORE_DECL	75
6.7.3	Typedef Documentation	75
6.7.3.1	binary_semaphore_t	75
6.7.4	Function Documentation	75
6.7.4.1	chBSemObjectInit(binary_semaphore_t *bsp, bool taken)	75
6.7.4.2	chBSemWait(binary_semaphore_t *bsp)	75
6.7.4.3	chBSemWaitS(binary_semaphore_t *bsp)	76
6.7.4.4	chBSemWaitTimeoutS(binary_semaphore_t *bsp, sysinterval_t timeout)	76
6.7.4.5	chBSemWaitTimeout(binary_semaphore_t *bsp, sysinterval_t timeout)	77
6.7.4.6	chBSemResetI(binary_semaphore_t *bsp, bool taken)	78

6.7.4.7	chBSemReset(binary_semaphore_t *bsp, bool taken)	79
6.7.4.8	chBSemSignall(binary_semaphore_t *bsp)	79
6.7.4.9	chBSemSignal(binary_semaphore_t *bsp)	80
6.7.4.10	chBSemGetStatel(const binary_semaphore_t *bsp)	81
6.8	Mailboxes	82
6.8.1	Detailed Description	82
6.8.2	Macro Definition Documentation	83
6.8.2.1	_MAILBOX_DATA	83
6.8.2.2	MAILBOX_DECL	84
6.8.3	Function Documentation	84
6.8.3.1	chMBOBJECTInit(mailbox_t *mbp, msg_t *buf, size_t n)	84
6.8.3.2	chMBReset(mailbox_t *mbp)	84
6.8.3.3	chMBResetl(mailbox_t *mbp)	85
6.8.3.4	chMBPostTimeout(mailbox_t *mbp, msg_t msg, sysinterval_t timeout)	85
6.8.3.5	chMBPostTimeoutS(mailbox_t *mbp, msg_t msg, sysinterval_t timeout)	86
6.8.3.6	chMBPostl(mailbox_t *mbp, msg_t msg)	87
6.8.3.7	chMBPostAheadTimeout(mailbox_t *mbp, msg_t msg, sysinterval_t timeout)	88
6.8.3.8	chMBPostAheadTimeoutS(mailbox_t *mbp, msg_t msg, sysinterval_t timeout)	89
6.8.3.9	chMBPostAheadl(mailbox_t *mbp, msg_t msg)	90
6.8.3.10	chMBFetchTimeout(mailbox_t *mbp, msg_t *msgp, sysinterval_t timeout)	91
6.8.3.11	chMBFetchTimeoutS(mailbox_t *mbp, msg_t *msgp, sysinterval_t timeout)	92
6.8.3.12	chMBFetchl(mailbox_t *mbp, msg_t *msgp)	93
6.8.3.13	chMBGetSize(const mailbox_t *mbp)	94
6.8.3.14	chMBGetUsedCountl(const mailbox_t *mbp)	94
6.8.3.15	chMBGetFreeCountl(const mailbox_t *mbp)	94
6.8.3.16	chMBPeekl(const mailbox_t *mbp)	95
6.8.3.17	chMBResumeX(mailbox_t *mbp)	96
6.9	Pipes	97
6.9.1	Detailed Description	97
6.9.2	Macro Definition Documentation	97
6.9.2.1	_PIPE_DATA	97
6.9.2.2	PIPE_DECL	98
6.9.3	Function Documentation	98
6.9.3.1	pipe_write(pipe_t *pp, const uint8_t *bp, size_t n)	98
6.9.3.2	pipe_read(pipe_t *pp, uint8_t *bp, size_t n)	99
6.9.3.3	chPipeObjectInit(pipe_t *pp, uint8_t *buf, size_t n)	100
6.9.3.4	chPipeReset(pipe_t *pp)	100
6.9.3.5	chPipeWriteTimeout(pipe_t *pp, const uint8_t *bp, size_t n, sysinterval_t timeout)	100
6.9.3.6	chPipeReadTimeout(pipe_t *pp, uint8_t *bp, size_t n, sysinterval_t timeout)	101
6.9.3.7	chPipeGetSize(const pipe_t *pp)	102

6.9.3.8	chPipeGetUsedCount(const pipe_t *pp)	102
6.9.3.9	chPipeGetFreeCount(const pipe_t *pp)	103
6.9.3.10	chPipeResume(pipe_t *pp)	103
6.10	Memory Management	104
6.10.1	Detailed Description	104
6.11	Core Memory Manager	105
6.11.1	Detailed Description	105
6.11.2	Macro Definition Documentation	106
6.11.2.1	CH_CFG_MEMCORE_SIZE	106
6.11.3	Typedef Documentation	106
6.11.3.1	memgetfunc_t	106
6.11.3.2	memgetfunc2_t	106
6.11.4	Function Documentation	106
6.11.4.1	_core_init(void)	106
6.11.4.2	chCoreAllocAlignedWithOffset(size_t size, unsigned align, size_t offset)	107
6.11.4.3	chCoreAllocAlignedWithOffset(size_t size, unsigned align, size_t offset)	107
6.11.4.4	chCoreGetStatusX(void)	108
6.11.4.5	chCoreAllocAligned(size_t size, unsigned align)	108
6.11.4.6	chCoreAllocAligned(size_t size, unsigned align)	109
6.11.4.7	chCoreAlloc(size_t size)	109
6.11.4.8	chCoreAlloc(size_t size)	110
6.11.5	Variable Documentation	111
6.11.5.1	ch_memcore	111
6.12	Memory Heaps	112
6.12.1	Detailed Description	112
6.12.2	Macro Definition Documentation	113
6.12.2.1	CH_HEAP_ALIGNMENT	113
6.12.2.2	CH_HEAP_AREA	113
6.12.3	Typedef Documentation	113
6.12.3.1	memory_heap_t	113
6.12.3.2	heap_header_t	113
6.12.4	Function Documentation	113
6.12.4.1	_heap_init(void)	113
6.12.4.2	chHeapObjectInit(memory_heap_t *heapp, void *buf, size_t size)	114
6.12.4.3	chHeapAllocAligned(memory_heap_t *heapp, size_t size, unsigned align)	114
6.12.4.4	chHeapFree(void *p)	115
6.12.4.5	chHeapStatus(memory_heap_t *heapp, size_t *totalp, size_t *largestp)	115
6.12.4.6	chHeapAlloc(memory_heap_t *heapp, size_t size)	115
6.12.4.7	chHeapGetSize(const void *p)	116
6.12.5	Variable Documentation	116

6.12.5.1	default_heap	116
6.13	Memory Pools	117
6.13.1	Detailed Description	117
6.13.2	Macro Definition Documentation	118
6.13.2.1	_MEMORYPOOL_DATA	118
6.13.2.2	MEMORYPOOL_DECL	119
6.13.2.3	_GUARDEDMEMORYPOOL_DATA	119
6.13.2.4	GUARDEDMEMORYPOOL_DECL	119
6.13.3	Function Documentation	119
6.13.3.1	chPoolObjectInitAligned(memory_pool_t *mp, size_t size, unsigned align, memgetfunc_t provider)	119
6.13.3.2	chPoolLoadArray(memory_pool_t *mp, void *p, size_t n)	120
6.13.3.3	chPoolAlloc(memory_pool_t *mp)	120
6.13.3.4	chPoolAlloc(memory_pool_t *mp)	121
6.13.3.5	chPoolFree(memory_pool_t *mp, void *objp)	122
6.13.3.6	chPoolFree(memory_pool_t *mp, void *objp)	122
6.13.3.7	chGuardedPoolObjectInitAligned(guarded_memory_pool_t *gmp, size_t size, unsigned align)	123
6.13.3.8	chGuardedPoolLoadArray(guarded_memory_pool_t *gmp, void *p, size_t n)	124
6.13.3.9	chGuardedPoolAllocTimeoutS(guarded_memory_pool_t *gmp, sysinterval_t timeout)	124
6.13.3.10	chGuardedPoolAllocTimeout(guarded_memory_pool_t *gmp, sysinterval_t timeout)	125
6.13.3.11	chGuardedPoolFree(guarded_memory_pool_t *gmp, void *objp)	126
6.13.3.12	chPoolObjectInit(memory_pool_t *mp, size_t size, memgetfunc_t provider)	126
6.13.3.13	chPoolAdd(memory_pool_t *mp, void *objp)	127
6.13.3.14	chPoolAddI(memory_pool_t *mp, void *objp)	128
6.13.3.15	chGuardedPoolObjectInit(guarded_memory_pool_t *gmp, size_t size)	128
6.13.3.16	chGuardedPoolGetCounterI(guarded_memory_pool_t *gmp)	129
6.13.3.17	chGuardedPoolAllocI(guarded_memory_pool_t *gmp)	129
6.13.3.18	chGuardedPoolFreeI(guarded_memory_pool_t *gmp, void *objp)	130
6.13.3.19	chGuardedPoolFreeS(guarded_memory_pool_t *gmp, void *objp)	131
6.13.3.20	chGuardedPoolAdd(guarded_memory_pool_t *gmp, void *objp)	132
6.13.3.21	chGuardedPoolAddI(guarded_memory_pool_t *gmp, void *objp)	132
6.13.3.22	chGuardedPoolAddS(guarded_memory_pool_t *gmp, void *objp)	133
6.14	Complex Services	134
6.14.1	Detailed Description	134
6.15	Objects FIFOs	135
6.15.1	Detailed Description	135
6.15.2	Typedef Documentation	136
6.15.2.1	objects_fifo_t	136

6.15.3	Function Documentation	136
6.15.3.1	chFifoObjectInitAligned(objects_fifo_t *ofp, size_t objsize, size_t objn, unsigned objalign, void *objbuf, msg_t *msgbuf)	136
6.15.3.2	chFifoObjectInit(objects_fifo_t *ofp, size_t objsize, size_t objn, void *objbuf, msg_t *msgbuf)	136
6.15.3.3	chFifoTakeObjectI(objects_fifo_t *ofp)	137
6.15.3.4	chFifoTakeObjectTimeoutS(objects_fifo_t *ofp, sysinterval_t timeout)	138
6.15.3.5	chFifoTakeObjectTimeout(objects_fifo_t *ofp, sysinterval_t timeout)	138
6.15.3.6	chFifoReturnObjectI(objects_fifo_t *ofp, void *objp)	139
6.15.3.7	chFifoReturnObjectS(objects_fifo_t *ofp, void *objp)	140
6.15.3.8	chFifoReturnObject(objects_fifo_t *ofp, void *objp)	140
6.15.3.9	chFifoSendObjectI(objects_fifo_t *ofp, void *objp)	141
6.15.3.10	chFifoSendObjectS(objects_fifo_t *ofp, void *objp)	141
6.15.3.11	chFifoSendObject(objects_fifo_t *ofp, void *objp)	142
6.15.3.12	chFifoSendObjectAheadI(objects_fifo_t *ofp, void *objp)	143
6.15.3.13	chFifoSendObjectAheadS(objects_fifo_t *ofp, void *objp)	143
6.15.3.14	chFifoSendObjectAhead(objects_fifo_t *ofp, void *objp)	144
6.15.3.15	chFifoReceiveObjectI(objects_fifo_t *ofp, void **objpp)	144
6.15.3.16	chFifoReceiveObjectTimeoutS(objects_fifo_t *ofp, void **objpp, sysinterval_t timeout)	145
6.15.3.17	chFifoReceiveObjectTimeout(objects_fifo_t *ofp, void **objpp, sysinterval_t timeout)	146
6.16	Dynamic Objects Factory	148
6.16.1	Detailed Description	148
6.16.2	Macro Definition Documentation	151
6.16.2.1	CH_CFG_FACTORY_MAX_NAMES_LENGTH	151
6.16.2.2	CH_CFG_FACTORY_OBJECTS_REGISTRY	151
6.16.2.3	CH_CFG_FACTORY_GENERIC_BUFFERS	151
6.16.2.4	CH_CFG_FACTORY_SEMAPHORES	151
6.16.2.5	CH_CFG_FACTORY_SEMAPHORES	151
6.16.2.6	CH_CFG_FACTORY_MAILBOXES	151
6.16.2.7	CH_CFG_FACTORY_MAILBOXES	151
6.16.2.8	CH_CFG_FACTORY_OBJ_FIFOS	151
6.16.2.9	CH_CFG_FACTORY_OBJ_FIFOS	151
6.16.2.10	CH_CFG_FACTORY_OBJ_FIFOS	151
6.16.2.11	CH_CFG_FACTORY_PIPES	152
6.16.2.12	CH_CFG_FACTORY_PIPES	152
6.16.3	Typedef Documentation	152
6.16.3.1	dyn_element_t	152
6.16.3.2	dyn_list_t	152
6.16.3.3	registered_object_t	152

6.16.3.4	<code>dyn_buffer_t</code>	152
6.16.3.5	<code>dyn_semaphore_t</code>	152
6.16.3.6	<code>dyn_mailbox_t</code>	152
6.16.3.7	<code>dyn_objects_fifo_t</code>	152
6.16.3.8	<code>dyn_pipe_t</code>	152
6.16.3.9	<code>objects_factory_t</code>	152
6.16.4	Function Documentation	153
6.16.4.1	<code>_factory_init(void)</code>	153
6.16.4.2	<code>chFactoryRegisterObject(const char *name, void *objp)</code>	153
6.16.4.3	<code>chFactoryFindObject(const char *name)</code>	153
6.16.4.4	<code>chFactoryFindObjectByPointer(void *objp)</code>	154
6.16.4.5	<code>chFactoryReleaseObject(registered_object_t *rop)</code>	154
6.16.4.6	<code>chFactoryCreateBuffer(const char *name, size_t size)</code>	155
6.16.4.7	<code>chFactoryFindBuffer(const char *name)</code>	155
6.16.4.8	<code>chFactoryReleaseBuffer(dyn_buffer_t *dbp)</code>	156
6.16.4.9	<code>chFactoryCreateSemaphore(const char *name, cnt_t n)</code>	156
6.16.4.10	<code>chFactoryFindSemaphore(const char *name)</code>	157
6.16.4.11	<code>chFactoryReleaseSemaphore(dyn_semaphore_t *dsp)</code>	157
6.16.4.12	<code>chFactoryCreateMailbox(const char *name, size_t n)</code>	157
6.16.4.13	<code>chFactoryFindMailbox(const char *name)</code>	158
6.16.4.14	<code>chFactoryReleaseMailbox(dyn_mailbox_t *dmp)</code>	159
6.16.4.15	<code>chFactoryCreateObjectsFIFO(const char *name, size_t objsize, size_t objn, unsigned objalign)</code>	159
6.16.4.16	<code>chFactoryFindObjectFIFO(const char *name)</code>	160
6.16.4.17	<code>chFactoryReleaseObjectsFIFO(dyn_objects_fifo_t *dofp)</code>	160
6.16.4.18	<code>chFactoryCreatePipe(const char *name, size_t size)</code>	161
6.16.4.19	<code>chFactoryFindPipe(const char *name)</code>	161
6.16.4.20	<code>chFactoryReleasePipe(dyn_pipe_t *dpp)</code>	162
6.16.4.21	<code>chFactoryDuplicateReference(dyn_element_t *dep)</code>	162
6.16.4.22	<code>chFactoryGetObject(registered_object_t *rop)</code>	162
6.16.4.23	<code>chFactoryGetBufferSize(dyn_buffer_t *dbp)</code>	163
6.16.4.24	<code>chFactoryGetBuffer(dyn_buffer_t *dbp)</code>	163
6.16.4.25	<code>chFactoryGetSemaphore(dyn_semaphore_t *dsp)</code>	164
6.16.4.26	<code>chFactoryGetMailbox(dyn_mailbox_t *dmp)</code>	164
6.16.4.27	<code>chFactoryGetObjectsFIFO(dyn_objects_fifo_t *dofp)</code>	164
6.16.4.28	<code>chFactoryGetPipe(dyn_pipe_t *dpp)</code>	165
6.16.5	Variable Documentation	165
6.16.5.1	<code>ch_factory</code>	165

7.1	ch_binary_semaphore Struct Reference	167
7.1.1	Detailed Description	168
7.2	ch_dyn_element Struct Reference	168
7.2.1	Detailed Description	169
7.2.2	Field Documentation	169
7.2.2.1	next	169
7.2.2.2	refs	169
7.3	ch_dyn_list Struct Reference	169
7.3.1	Detailed Description	169
7.4	ch_dyn_mailbox Struct Reference	170
7.4.1	Detailed Description	170
7.4.2	Field Documentation	171
7.4.2.1	element	171
7.4.2.2	mbx	171
7.4.2.3	msgbuf	171
7.5	ch_dyn_object Struct Reference	171
7.5.1	Detailed Description	172
7.5.2	Field Documentation	172
7.5.2.1	element	172
7.5.2.2	buffer	172
7.6	ch_dyn_objects_fifo Struct Reference	172
7.6.1	Detailed Description	173
7.6.2	Field Documentation	173
7.6.2.1	element	173
7.6.2.2	fifo	174
7.6.2.3	msgbuf	174
7.7	ch_dyn_pipe Struct Reference	174
7.7.1	Detailed Description	175
7.7.2	Field Documentation	176
7.7.2.1	element	176
7.7.2.2	pipe	176
7.7.2.3	buffer	176
7.8	ch_dyn_semaphore Struct Reference	176
7.8.1	Detailed Description	177
7.8.2	Field Documentation	177
7.8.2.1	element	177
7.8.2.2	sem	177
7.9	ch_objects_factory Struct Reference	177
7.9.1	Detailed Description	178
7.9.2	Field Documentation	178

7.9.2.1	mtx	178
7.9.2.2	obj_list	178
7.9.2.3	obj_pool	178
7.9.2.4	buf_list	178
7.9.2.5	sem_list	178
7.9.2.6	sem_pool	178
7.9.2.7	mbx_list	179
7.9.2.8	fifo_list	179
7.9.2.9	pipe_list	179
7.10	ch_objects_fifo Struct Reference	179
7.10.1	Detailed Description	180
7.10.2	Field Documentation	180
7.10.2.1	free	180
7.10.2.2	mbx	180
7.11	ch_registered_static_object Struct Reference	180
7.11.1	Detailed Description	181
7.11.2	Field Documentation	181
7.11.2.1	element	181
7.11.2.2	objp	181
7.12	guarded_memory_pool_t Struct Reference	181
7.12.1	Detailed Description	182
7.12.2	Field Documentation	182
7.12.2.1	sem	182
7.12.2.2	pool	182
7.13	heap_header Union Reference	183
7.13.1	Detailed Description	183
7.13.2	Field Documentation	183
7.13.2.1	next	183
7.13.2.2	pages	183
7.13.2.3	heap	183
7.13.2.4	size	183
7.14	mailbox_t Struct Reference	184
7.14.1	Detailed Description	184
7.14.2	Field Documentation	185
7.14.2.1	buffer	185
7.14.2.2	top	185
7.14.2.3	wrptr	185
7.14.2.4	rdptr	185
7.14.2.5	cnt	185
7.14.2.6	reset	185

7.14.2.7	qw	185
7.14.2.8	qr	185
7.15	memcore_t Struct Reference	185
7.15.1	Detailed Description	186
7.15.2	Field Documentation	186
7.15.2.1	nextmem	186
7.15.2.2	endmem	186
7.16	memory_heap Struct Reference	186
7.16.1	Detailed Description	187
7.16.2	Field Documentation	187
7.16.2.1	provider	187
7.16.2.2	header	187
7.16.2.3	mtx	187
7.17	memory_pool_t Struct Reference	188
7.17.1	Detailed Description	188
7.17.2	Field Documentation	188
7.17.2.1	next	188
7.17.2.2	object_size	188
7.17.2.3	align	189
7.17.2.4	provider	189
7.18	nil_system Struct Reference	189
7.18.1	Detailed Description	190
7.18.2	Field Documentation	190
7.18.2.1	current	190
7.18.2.2	next	190
7.18.2.3	systeme	190
7.18.2.4	lasttime	190
7.18.2.5	nexttime	191
7.18.2.6	isr_cnt	191
7.18.2.7	lock_cnt	191
7.18.2.8	dbg_panic_msg	191
7.18.2.9	threads	191
7.19	nil_thread Struct Reference	191
7.19.1	Detailed Description	193
7.19.2	Field Documentation	193
7.19.2.1	ctx	193
7.19.2.2	state	193
7.19.2.3	msg	193
7.19.2.4	p	193
7.19.2.5	trp	193

7.19.2.6	tqp	193
7.19.2.7	semp	193
7.19.2.8	ewmask	193
7.19.2.9	timeout	193
7.19.2.10	epmask	193
7.19.2.11	wabase	193
7.20	nil_thread_cfg Struct Reference	194
7.20.1	Detailed Description	194
7.20.2	Field Documentation	194
7.20.2.1	wbase	194
7.20.2.2	wend	194
7.20.2.3	namep	194
7.20.2.4	funcp	195
7.20.2.5	arg	195
7.21	nil_threads_queue Struct Reference	195
7.21.1	Detailed Description	196
7.21.2	Field Documentation	196
7.21.2.1	cnt	196
7.22	pipe_t Struct Reference	196
7.22.1	Detailed Description	198
7.22.2	Field Documentation	198
7.22.2.1	buffer	198
7.22.2.2	top	198
7.22.2.3	wrptr	198
7.22.2.4	rdptr	198
7.22.2.5	cnt	198
7.22.2.6	reset	198
7.22.2.7	wtr	198
7.22.2.8	rtr	199
7.22.2.9	cmtx	199
7.22.2.10	wmtx	199
7.22.2.11	rmtx	199
7.23	pool_header Struct Reference	199
7.23.1	Detailed Description	199
7.23.2	Field Documentation	199
7.23.2.1	next	199
8	File Documentation	201
8.1	ch.c File Reference	201
8.1.1	Detailed Description	203

8.2	ch.h File Reference	203
8.2.1	Detailed Description	209
8.3	chbsem.h File Reference	209
8.3.1	Detailed Description	210
8.4	chconf.h File Reference	211
8.4.1	Detailed Description	212
8.5	chfactory.c File Reference	212
8.5.1	Detailed Description	214
8.6	chfactory.h File Reference	214
8.6.1	Detailed Description	216
8.7	chlib.h File Reference	216
8.7.1	Detailed Description	217
8.8	chmboxes.c File Reference	217
8.8.1	Detailed Description	218
8.9	chmboxes.h File Reference	218
8.9.1	Detailed Description	219
8.10	chmemcore.c File Reference	219
8.10.1	Detailed Description	220
8.11	chmemcore.h File Reference	220
8.11.1	Detailed Description	220
8.12	chmemheaps.c File Reference	221
8.12.1	Detailed Description	221
8.13	chmemheaps.h File Reference	221
8.13.1	Detailed Description	222
8.14	chmempools.c File Reference	222
8.14.1	Detailed Description	223
8.15	chmempools.h File Reference	223
8.15.1	Detailed Description	224
8.16	chobjfifos.h File Reference	224
8.16.1	Detailed Description	225
8.17	chpipes.c File Reference	226
8.17.1	Detailed Description	226
8.18	chpipes.h File Reference	227
8.18.1	Detailed Description	227

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
- OS Library 71
 - Version Numbers and Identification 72
 - Synchronization 73
 - Binary Semaphores 74
 - Mailboxes 82
 - Pipes 97
 - Memory Management 104
 - Core Memory Manager 105
 - Memory Heaps 112
 - Memory Pools 117
 - Complex Services 134
 - Objects FIFOs 135
 - Dynamic Objects Factory 148

Chapter 3

Hierarchical Index

3.1 Class Hierarchy

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

ch_dyn_element	168
ch_dyn_list	169
ch_dyn_mailbox	170
ch_dyn_object	171
ch_dyn_objects_fifo	172
ch_dyn_pipe	174
ch_dyn_semaphore	176
ch_objects_factory	177
ch_objects_fifo	179
ch_registered_static_object	180
guarded_memory_pool_t	181
heap_header	183
mailbox_t	184
memcore_t	185
memory_heap	186
memory_pool_t	188
nil_system	189
nil_thread	191
nil_thread_cfg	194
nil_threads_queue	195
ch_binary_semaphore	167
pipe_t	196
pool_header	199

Chapter 4

Data Structure Index

4.1 Data Structures

Here are the data structures with brief descriptions:

ch_binary_semaphore	Binary semaphore type	167
ch_dyn_element	Type of a dynamic object list element	168
ch_dyn_list	Type of a dynamic object list	169
ch_dyn_mailbox	Type of a dynamic buffer object	170
ch_dyn_object	Type of a dynamic buffer object	171
ch_dyn_objects_fifo	Type of a dynamic buffer object	172
ch_dyn_pipe	Type of a dynamic pipe object	174
ch_dyn_semaphore	Type of a dynamic semaphore	176
ch_objects_factory	Type of the factory main object	177
ch_objects_fifo	Type of an objects FIFO	179
ch_registered_static_object	Type of a registered object	180
guarded_memory_pool_t	Guarded memory pool descriptor	181
heap_header	Memory heap block header	183
mailbox_t	Structure representing a mailbox object	184
memcore_t	Type of memory core object	185
memory_heap	Structure describing a memory heap	186
memory_pool_t	Memory pool descriptor	188
nil_system	System data structure	189
nil_thread	Structure representing a thread	191

nil_thread_cfg	Structure representing a thread static configuration	194
nil_threads_queue	Structure representing a queue of threads	195
pipe_t	Structure representing a pipe object	196
pool_header	Memory pool free object header	199

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	201
ch.h	Nil RTOS main header file	203
chbsem.h	Binary semaphores structures and macros	209
chconf.h	Configuration file template	211
chfactory.c	ChibiOS objects factory and registry code	212
chfactory.h	ChibiOS objects factory structures and macros	214
chlib.h	ChibiOS/LIB main include file	216
chmboxes.c	Mailboxes code	217
chmboxes.h	Mailboxes macros and structures	218
chmemcore.c	Core memory manager code	219
chmemcore.h	Core memory manager macros and structures	220
chmemheaps.c	Memory heaps code	221
chmemheaps.h	Memory heaps macros and structures	221
chmempools.c	Memory Pools code	222
chmempools.h	Memory Pools macros and structures	223
chobjfifos.h	Objects FIFO structures and macros	224
chpipes.c	Pipes code	226
chpipes.h	Pipes macros and structures	227

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_USE_PIPES` TRUE
Pipes 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.
- #define CH_CFG_FACTORY_PIPES TRUE

Enables factory for Pipes.

Debug options

- #define CH_DBG_STATISTICS FALSE

Debug option, kernel statistics.
- #define CH_DBG_SYSTEM_STATE_CHECK TRUE

Debug option, system state check.
- #define CH_DBG_ENABLE_CHECKS TRUE

Debug option, parameters checks.
- #define CH_DBG_ENABLE_ASSERTS TRUE

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_USE_PIPES TRUE

Pipes APIs.

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

Note

The default is `TRUE`.

6.2.2.14 `#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`.

6.2.2.15 `#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`.

6.2.2.16 `#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.

6.2.2.17 `#define CH_CFG_FACTORY_OBJECTS_REGISTRY TRUE`

Enables the registry of generic objects.

6.2.2.18 `#define CH_CFG_FACTORY_GENERIC_BUFFERS TRUE`

Enables factory for generic buffers.

6.2.2.19 `#define CH_CFG_FACTORY_SEMAPHORES TRUE`

Enables factory for semaphores.

6.2.2.20 `#define CH_CFG_FACTORY_MAILBOXES TRUE`

Enables factory for mailboxes.

6.2.2.21 `#define CH_CFG_FACTORY_OBJ_FIFOS TRUE`

Enables factory for objects FIFOs.

6.2.2.22 `#define CH_CFG_FACTORY_PIPES TRUE`

Enables factory for Pipes.

6.2.2.23 #define CH_DBG_STATISTICS FALSE

Debug option, kernel statistics.

Note

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

6.2.2.24 #define CH_DBG_SYSTEM_STATE_CHECK TRUE

Debug option, system state check.

Note

The default is `FALSE`.

6.2.2.25 #define CH_DBG_ENABLE_CHECKS TRUE

Debug option, parameters checks.

Note

The default is `FALSE`.

6.2.2.26 #define CH_DBG_ENABLE_ASSERTS TRUE

System assertions.

Note

The default is `FALSE`.

6.2.2.27 #define CH_DBG_ENABLE_STACK_CHECK TRUE

Stack check.

Note

The default is `FALSE`.

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

```
{  
    \  
}
```

System initialization hook.

6.2.2.29 #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.

6.2.2.30 #define CH_CFG_THREAD_EXT_INIT_HOOK(*tr*)

Value:

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

Threads initialization hook.

6.2.2.31 #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.

6.2.2.32 #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.

6.2.2.33 #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 "3.2.2"`
Kernel version string.
- `#define CH_KERNEL_MAJOR 3`
Kernel version major number.
- `#define CH_KERNEL_MINOR 2`
Kernel version minor number.
- `#define CH_KERNEL_PATCH 2`
Kernel version patch number.

Constants for configuration options

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

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`(tp) ((tp)->state == `NIL_STATE_READY`)
- #define `NIL_THD_IS_SLEEPING`(tp) ((tp)->state == `NIL_STATE_SLEEPING`)
- #define `NIL_THD_IS_SUSP`(tp) ((tp)->state == `NIL_STATE_SUSP`)
- #define `NIL_THD_IS_WTQUEUE`(tp) ((tp)->state == `NIL_STATE_WTQUEUE`)
- #define `NIL_THD_IS_WTOREVT`(tp) ((tp)->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 `chThdQueueIsEmpty`(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 `chSemFastWaitl`(sp) ((sp)->cnt--)
Decreases the semaphore counter.
- #define `chSemFastSignal`(sp) ((sp)->cnt++)
Increases the semaphore counter.
- #define `chSemGetCounterl`(sp) ((sp)->cnt)
Returns the semaphore counter current value.
- #define `chVTGetSystemTimeX`() (nil.systemtime)
Current system time.
- #define `chVTTimeElapsedSinceX`(start) `chTimeDiffX`((start), `chVTGetSystemTimeX`())
Returns the elapsed time since the specified start time.
- #define `chTimeAddX`(systemtime, interval) ((`systemtime_t`)(systemtime) + (`systemtime_t`)(interval))
Adds an interval to a system time returning a system time.
- #define `chTimeDiffX`(start, end) ((`sysinterval_t`)((`systemtime_t`)(end) - (`systemtime_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 `systemtime_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

- static [thread_t](#) * [nil_find_thread](#) ([tstate_t](#) state, void *p)
Retrieves the highest priority thread in the specified state and associated to the specified object.
- static [cnt_t](#) [nil_ready_all](#) (void *p, [cnt_t](#) cnt, [msg_t](#) msg)
Puts in ready state all thread matching the specified status and associated object.
- 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 `chThdResume` (`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` (`system_time_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 `chSemResetl` (`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 `chEvtSignal1` (`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 "3.2.2"`

Kernel version string.

6.3.2.4 `#define CH_KERNEL_MAJOR 3`

Kernel version major number.

6.3.2.5 `#define CH_KERNEL_MINOR 2`

Kernel version minor number.

6.3.2.6 `#define CH_KERNEL_PATCH 2`

Kernel version patch number.

6.3.2.7 `#define FALSE 0`

Generic 'false' preprocessor boolean constant.

Note

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

6.3.2.8 #define TRUE 1

Generic 'true' preprocessor boolean constant.

Note

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

6.3.2.9 #define MSG_OK (msg_t)0

OK wakeup message.

6.3.2.10 #define MSG_TIMEOUT (msg_t)-1

Wake-up caused by a timeout condition.

6.3.2.11 #define MSG_RESET (msg_t)-2

Wake-up caused by a reset condition.

6.3.2.12 #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.

6.3.2.13 #define TIME_INFINITE ((sysinterval_t)0)

Infinite time specification for all functions with a timeout specification.

6.3.2.14 #define TIME_MAX_INTERVAL ((sysinterval_t)-2)

Maximum interval constant usable as timeout.

6.3.2.15 #define TIME_MAX_SYSTIME ((systime_t)-1)

Maximum system of system time before it wraps.

6.3.2.16 #define NIL_STATE_READY (tstate_t)0

Thread ready or executing.

6.3.2.17 #define NIL_STATE_SLEEPING (tstate_t)1

Thread sleeping.

6.3.2.18 #define NIL_STATE_SUSP (tstate_t)2

Thread suspended.

6.3.2.19 #define NIL_STATE_WTQUEUE (tstate_t)3

On queue or semaph.

6.3.2.20 #define NIL_STATE_WTOREVT (tstate_t)4

Waiting for events.

6.3.2.21 #define ALL_EVENTS ((eventmask_t)-1)

All events allowed mask.

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

Returns an event mask from an event identifier.

6.3.2.23 #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`.

6.3.2.24 #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.

6.3.2.25 #define CH_CFG_FACTORY_OBJECTS_REGISTRY TRUE

Enables the registry of generic objects.

6.3.2.26 #define CH_CFG_FACTORY_GENERIC_BUFFERS TRUE

Enables factory for generic buffers.

6.3.2.27 #define CH_CFG_FACTORY_SEMAPHORES TRUE

Enables factory for semaphores.

6.3.2.28 #define CH_CFG_FACTORY_MAILBOXES TRUE

Enables factory for mailboxes.

6.3.2.29 #define CH_CFG_FACTORY_OBJ_FIFOS TRUE

Enables factory for objects FIFOs.

6.3.2.30 #define THD_IDLE_BASE (&__main_thread_stack_base__)

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

6.3.2.31 #define __CH_STRINGIFY(a) #a

Utility to make the parameter a quoted string.

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

Start of user threads table.

6.3.2.33 #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.

6.3.2.34 #define THD_TABLE_END

Value:

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

End of user threads table.

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

Alignment mask constant.

Parameters

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

6.3.2.36 #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

6.3.2.37 #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

```
6.3.2.38 #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

```
6.3.2.39 #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
----	----------	---

```
6.3.2.40 #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.

6.3.2.41 `#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.

6.3.2.42 `#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.

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

Priority level validation macro.

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

Parameters

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

Returns

Priority range result.

Return values

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

6.3.2.44 `#define CH_IRQ_IS_VALID_KERNEL_PRIORITY(prio) PORT_IRQ_IS_VALID_KERNEL_PRIORITY(prio)`

Priority level validation macro.

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

Parameters

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

Returns

Priority range result.

Return values

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

6.3.2.45 #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.

6.3.2.46 #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.

6.3.2.47 #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.

6.3.2.48 #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.

6.3.2.49 #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

<i>in</i>	<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.

6.3.2.50 #define TIME_MS2I(*msecs*)

Value:

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

Milliseconds to time interval.

Converts from milliseconds to system ticks number.

Note

The result is rounded upward to the next tick boundary.
Use of this macro for large values is not secure because integer overflows, make sure your value can be correctly converted.

Parameters

in	<i>msecs</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.

6.3.2.51 #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.

6.3.2.52 #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.

6.3.2.53 #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.

6.3.2.54 #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.

6.3.2.55 `#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
----	-------------	--

6.3.2.56 `#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
----	-------------	--

6.3.2.57 `#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

6.3.2.58 `#define SEMAPHORE_DECL(name, n) semaphore_t name = _SEMAPHORE_DATA(name, n)`

Static semaphore initializer.

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

Parameters

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

6.3.2.59 `#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.

6.3.2.60 `#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.

6.3.2.61 `#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.

6.3.2.62 #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.

6.3.2.63 #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.

6.3.2.64 #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.

6.3.2.65 #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.

6.3.2.66 #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.

6.3.2.67 #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.

6.3.2.68 #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.

6.3.2.69 #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.

6.3.2.70 #define chThdSleepMilliseconds(msec) chThdSleep(TIME_MS2I(msecs))

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>msecs</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.

6.3.2.71 #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.

6.3.2.72 #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.

6.3.2.73 #define chThdSleepUntilS(*abstime*)

Value:

```
(void) chSchGoSleepTimeoutS(NIL_STATE_SLEEPING,
    \
    chTimeDiffX(chVGetSystemTimeX(), (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.

6.3.2.74 #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.

6.3.2.75 `#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.

6.3.2.76 `#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.

6.3.2.77 `#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

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

Function Class:

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

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

Performs a wait operation on a semaphore.

Parameters

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

Returns

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

Return values

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

Function Class:

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

6.3.2.79 #define chSemFastWaitI(sp) ((sp)->cnt--)

Decreases the semaphore counter.

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

Parameters

in	<code>sp</code>	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.

6.3.2.80 `#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	<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.

6.3.2.81 `#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.

6.3.2.82 `#define chVTGetSystemTimeX() (nil.systemtime)`

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 `systemtime_t` size.

Returns

The system time in ticks.

Function Class:

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

6.3.2.83 `#define chVTimeElapsedSinceX(start) chTimeDiffX((start), chVTGetSystemTimeX())`

Returns the elapsed time since the specified start time.

Parameters

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

Returns

The elapsed time.

Function Class:

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

```
6.3.2.84 #define chTimeAddX( systeme, interval ) ((systeme_t)(systeme) + (systeme_t)(interval))
```

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

Parameters

in	<i>systeme</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.

```
6.3.2.85 #define chTimeDiffX( start, end ) ((sysinterval_t)((systeme_t)((systeme_t)(end) - (systeme_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.

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

Value:

```
((bool)((systeme_t)((systeme_t)(time) - (systeme_t)(start)) < \
      (systeme_t)((systeme_t)(end) - (systeme_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.

6.3.2.87 #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.

6.3.2.88 #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 static thread_t* nil_find_thread (tstate_t state, void * p) [static]**

Retrieves the highest priority thread in the specified state and associated to the specified object.

Note

The search is unbounded, the thread is assumed to exist.

Parameters

in	<i>state</i>	thread state
in	<i>p</i>	object pointer

Returns

Pointer to the thread.

6.3.4.2 static cnt_t nil_ready_all (void * p, cnt_t cnt, msg_t msg) [static]

Puts in ready state all thread matching the specified status and associated object.

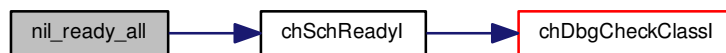
Parameters

in	<i>p</i>	object pointer
in	<i>cnt</i>	number of threads to be readied as a negative number, non negative numbers are ignored
in	<i>msg</i>	the wakeup message

Returns

Always zero.

Here is the call graph for this function:

6.3.4.3 `void _dbg_check_disable (void)`

Guard code for `chSysDisable ()`.

Function Class:

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

Here is the call graph for this function:

6.3.4.4 `void _dbg_check_suspend (void)`

Guard code for `chSysSuspend ()`.

Function Class:

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

Here is the call graph for this function:



6.3.4.5 void _dbg_check_enable (void)

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

Function Class:

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

Here is the call graph for this function:



6.3.4.6 void _dbg_check_lock (void)

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

Function Class:

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

Here is the call graph for this function:



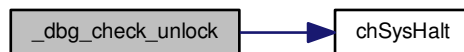
6.3.4.7 void _dbg_check_unlock (void)

Guard code for `chSysUnlock()`.

Function Class:

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

Here is the call graph for this function:



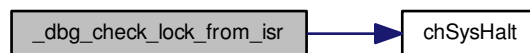
6.3.4.8 void _dbg_check_lock_from_isr (void)

Guard code for `chSysLockFromIsr()`.

Function Class:

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

Here is the call graph for this function:



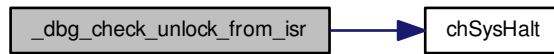
6.3.4.9 void _dbg_check_unlock_from_isr (void)

Guard code for `chSysUnlockFromIsr()`.

Function Class:

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

Here is the call graph for this function:



6.3.4.10 void _dbg_check_enter_isr (void)

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

Function Class:

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

Here is the call graph for this function:



6.3.4.11 void _dbg_check_leave_isr (void)

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

Function Class:

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

Here is the call graph for this function:



6.3.4.12 void chDbgCheckClassI (void)

I-class functions context check.

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

Function Class:

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

Here is the call graph for this function:



6.3.4.13 void chDbgCheckClassS (void)

S-class functions context check.

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

Function Class:

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

Here is the call graph for this function:



6.3.4.14 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.

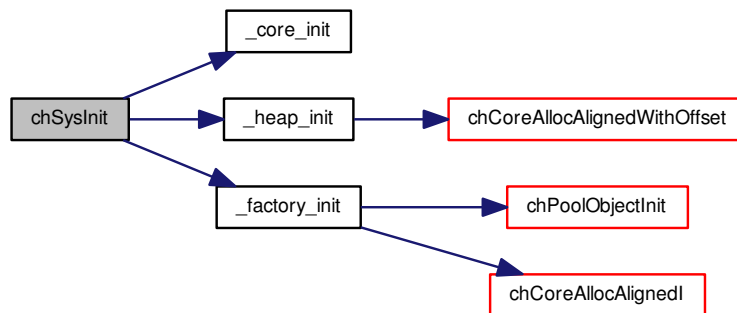
Note

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:

**6.3.4.15 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.

6.3.4.16 void chSysTimerHandlerl (void)

Time management handler.

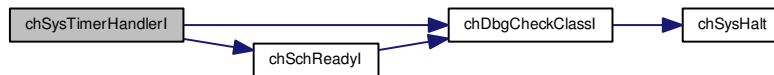
Note

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:

**6.3.4.17 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.

6.3.4.18 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.

6.3.4.19 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.

6.3.4.20 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.

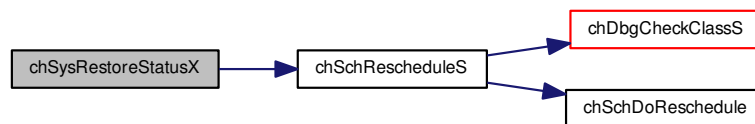
Parameters

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:

**6.3.4.21 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	cnt	the counter value to be tested
in	start	the start of the time window (inclusive)
in	end	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.

6.3.4.22 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:

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

Makes the specified thread ready for execution.

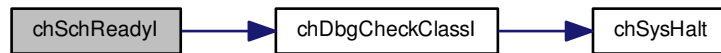
Parameters

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:

**6.3.4.24 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.

6.3.4.25 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.

6.3.4.26 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:



6.3.4.27 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.

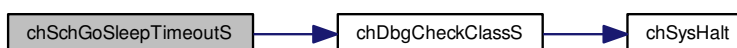
Return values

<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:



6.3.4.28 `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. • <i>TIME_IMMEDIATE</i> immediate timeout.

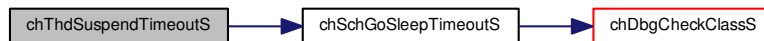
Returns

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:



6.3.4.29 `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.

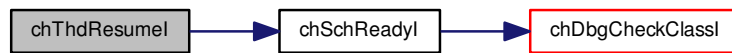
Parameters

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:



6.3.4.30 void chThdResume (thread_reference_t * trp, msg_t msg)

Wakes up a thread waiting on a thread reference object.

Note

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

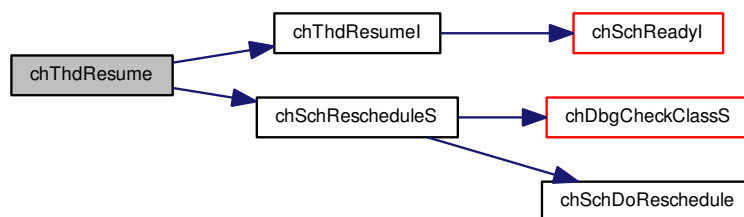
Parameters

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

Function Class:

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

Here is the call graph for this function:



6.3.4.31 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.

6.3.4.32 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.

6.3.4.33 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> no timeout. • <i>TIME_IMMEDIATE</i> immediate timeout.

Returns

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

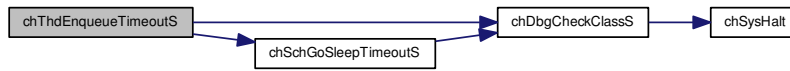
Return values

<i>MSG_TIMEOUT</i>	if the thread has not been dequeued within the specified timeout or if the function has been invoked with <i>TIME_IMMEDIATE</i> 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:



6.3.4.34 void chThdDoDequeueNextl (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.

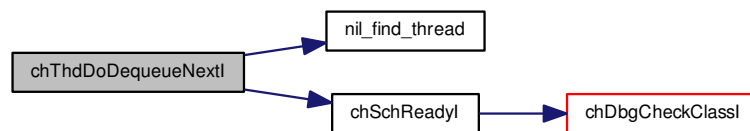
Parameters

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:



6.3.4.35 void chThdDequeueNextl (threads_queue_t * *tqp*, msg_t *msg*)

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

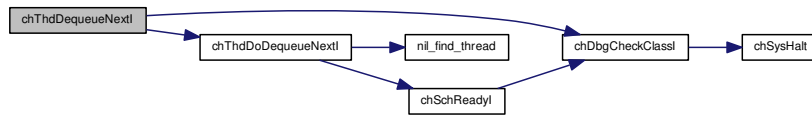
Parameters

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:



6.3.4.36 void chThdDequeueAlll (threads_queue_t * tqp, msg_t msg)

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

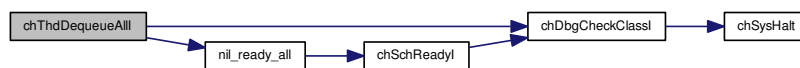
Parameters

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:



6.3.4.37 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 semaphore_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_INFINITE</i> no timeout. • <i>TIME_IMMEDIATE</i> immediate timeout.

Returns

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

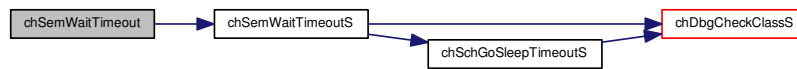
Return values

<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:

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

Here is the call graph for this function:

**6.3.4.38 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"> • <i>TIME_INFINITE</i> no timeout. • <i>TIME_IMMEDIATE</i> immediate timeout.

Returns

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

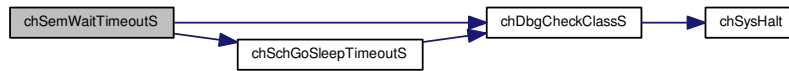
Return values

<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:



6.3.4.39 void chSemSignal (semaphore_t * sp)

Performs a signal operation on a semaphore.

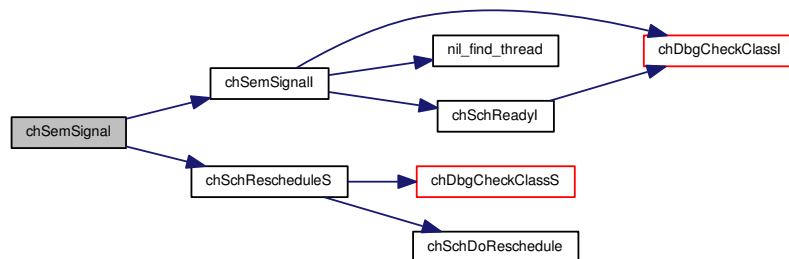
Parameters

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:



6.3.4.40 void chSemSignal (semaphore_t * sp)

Performs a signal operation on a semaphore.

Postcondition

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

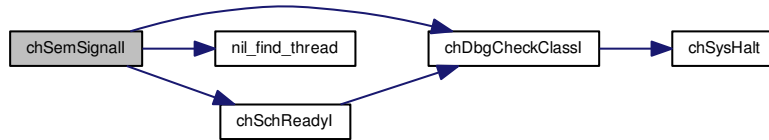
Parameters

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

Function Class:

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

Here is the call graph for this function:



6.3.4.41 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.

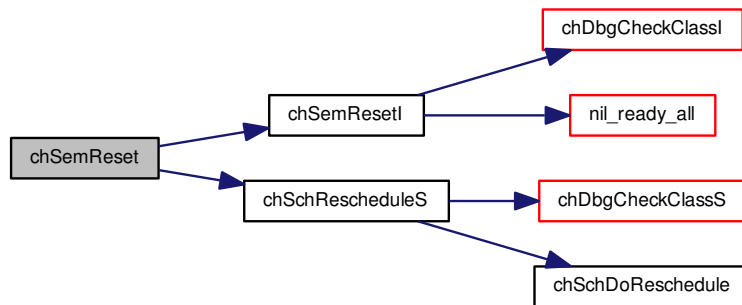
Parameters

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



6.3.4.42 void chSemResetl (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.

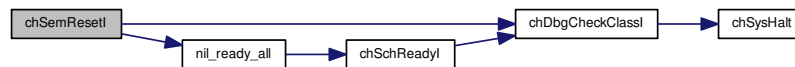
Parameters

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

Function Class:

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

Here is the call graph for this function:



6.3.4.43 void chEvtSignal (thread_t * tp, eventmask_t mask)

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

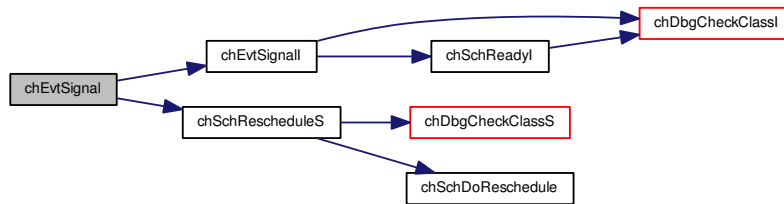
Parameters

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:



6.3.4.44 void chEvtSignal (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.

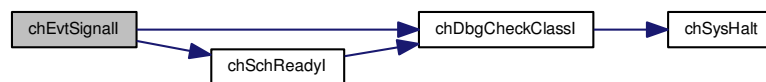
Parameters

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:



6.3.4.45 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_INFINITE</code> no timeout. • <code>TIME_IMMEDIATE</code> immediate timeout.

Returns

The mask of the served and cleared events.

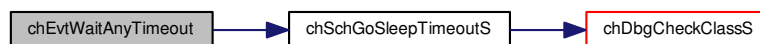
Return values

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 OS Library

6.4.1 Detailed Description

The OS Library is a set of RTOS extensions compatible with both the RT and NIL RTOSes.

Modules

- [Version Numbers and Identification](#)
- [Synchronization](#)
- [Memory Management](#)
- [Complex Services](#)

6.5 Version Numbers and Identification

6.5.1 Detailed Description

OS Library related info.

Macros

- `#define _CHIBIOS_OSLIB_`
ChibiOS/LIB identification macro.
- `#define CH_OSLIB_STABLE 1`
Stable release flag.

ChibiOS/LIB version identification

- `#define CH_OSLIB_VERSION "1.1.2"`
OS Library version string.
- `#define CH_OSLIB_MAJOR 1`
OS Library version major number.
- `#define CH_OSLIB_MINOR 1`
OS Library version minor number.
- `#define CH_OSLIB_PATCH 2`
OS Library version patch number.

6.5.2 Macro Definition Documentation

6.5.2.1 `#define _CHIBIOS_OSLIB_`

ChibiOS/LIB identification macro.

6.5.2.2 `#define CH_OSLIB_STABLE 1`

Stable release flag.

6.5.2.3 `#define CH_OSLIB_VERSION "1.1.2"`

OS Library version string.

6.5.2.4 `#define CH_OSLIB_MAJOR 1`

OS Library version major number.

6.5.2.5 `#define CH_OSLIB_MINOR 1`

OS Library version minor number.

6.5.2.6 `#define CH_OSLIB_PATCH 2`

OS Library version patch number.

6.6 Synchronization

6.6.1 Detailed Description

Synchronization services.

Modules

- [Binary Semaphores](#)
- [Mailboxes](#)
- [Pipes](#)

6.7 Binary Semaphores

6.7.1 Detailed Description

Macros

- `#define _BSEMAPHORE_DATA(name, taken) {_SEMAPHORE_DATA(name.sem, ((taken) ? 0 : 1))}`
Data part of a static semaphore initializer.
- `#define BSEMAPHORE_DECL(name, taken) binary_semaphore_t name = _BSEMAPHORE_DATA(name, taken)`
Static semaphore initializer.

Typedefs

- `typedef struct ch_binary_semaphore binary_semaphore_t`
Binary semaphore type.

Data Structures

- `struct ch_binary_semaphore`
Binary semaphore type.

Functions

- `static void chBSemObjectInit (binary_semaphore_t *bsp, bool taken)`
Initializes a binary semaphore.
- `static msg_t chBSemWait (binary_semaphore_t *bsp)`
Wait operation on the binary semaphore.
- `static msg_t chBSemWaitS (binary_semaphore_t *bsp)`
Wait operation on the binary semaphore.
- `static msg_t 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.

6.7.2 Macro Definition Documentation

6.7.2.1 `#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

6.7.2.2 `#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

6.7.3 Typedef Documentation

6.7.3.1 `typedef struct ch_binary_semaphore binary_semaphore_t`

Binary semaphore type.

6.7.4 Function Documentation

6.7.4.1 `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.

6.7.4.2 `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.

6.7.4.3 `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.

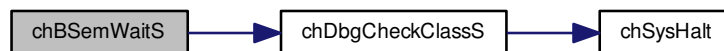
Return values

<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:



6.7.4.4 `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.

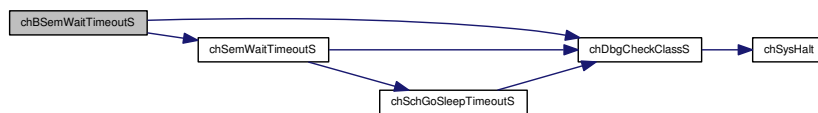
Return values

<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:



```

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

```

Wait operation on the binary semaphore.

Parameters

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

Returns

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

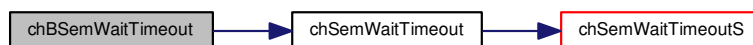
Return values

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

Function Class:

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

Here is the call graph for this function:



6.7.4.6 `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 `bsemWait()` will return `MSG_RESET` instead of `MSG_OK`.

This function does not reschedule.

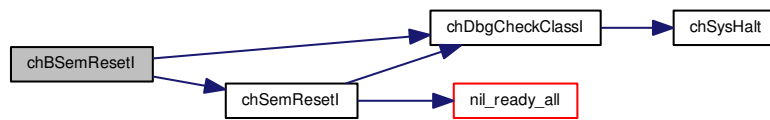
Parameters

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

Function Class:

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

Here is the call graph for this function:



6.7.4.7 `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`.

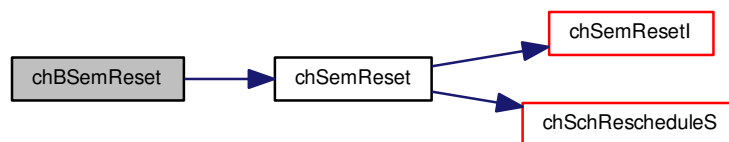
Parameters

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:



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

Performs a signal operation on a binary semaphore.

Note

This function does not reschedule.

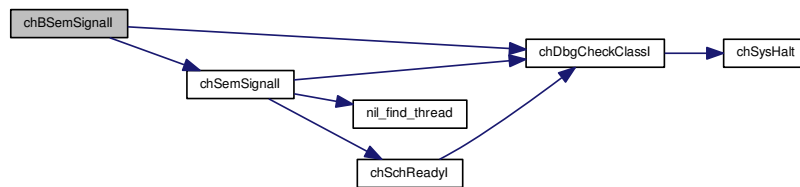
Parameters

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:



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

Performs a signal operation on a binary semaphore.

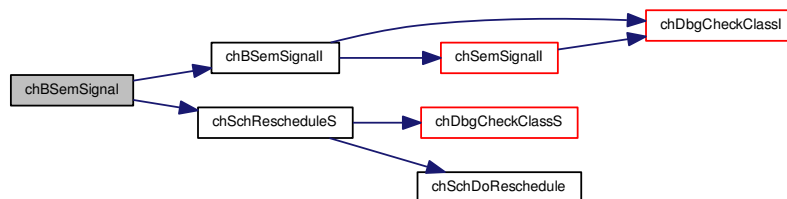
Parameters

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:



6.7.4.10 `static bool chBSemGetStateI (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.

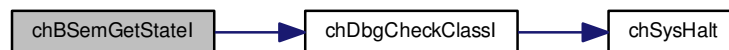
Return values

<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:



6.8 Mailboxes

6.8.1 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.*

6.8.2 Macro Definition Documentation

6.8.2.1 #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

6.8.2.2 `#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

6.8.3 Function Documentation

6.8.3.1 `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.

6.8.3.2 `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()`.

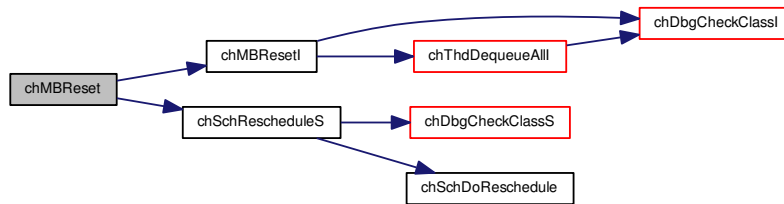
Parameters

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:



6.8.3.3 void chMResetI (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 `chMResumeX()`.

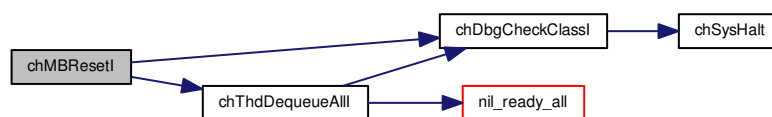
Parameters

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

Function Class:

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

Here is the call graph for this function:



6.8.3.4 msg_t chMPostTimeout (mailbox_t * mbp, msg_t msg, sysinterval_t timeout)

Posts a message into a mailbox.

The invoking thread waits until a empty slot in the mailbox becomes available or the specified time runs out.

Parameters

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

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.

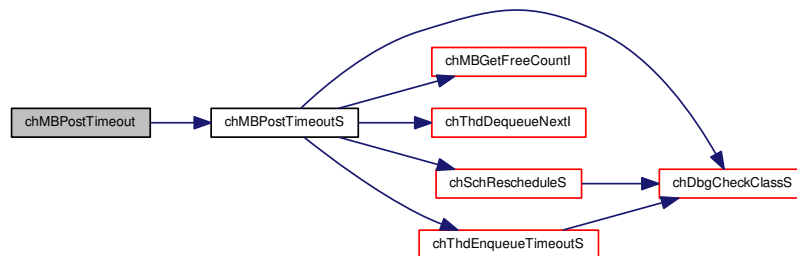
Return values

<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:

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

Returns

The operation status.

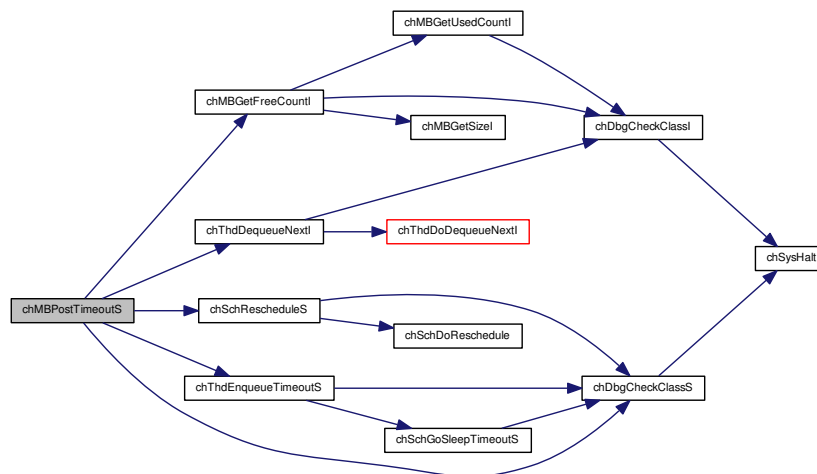
Return values

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

**6.8.3.6 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.
---------------	---

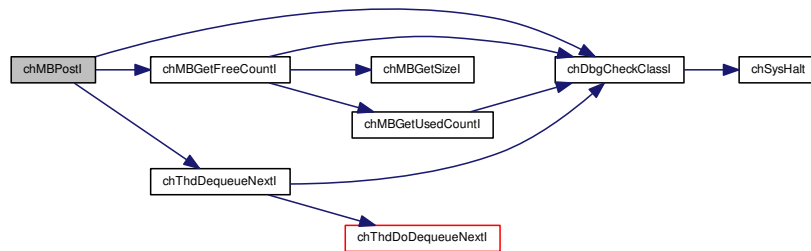
Return values

<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:



6.8.3.7 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.

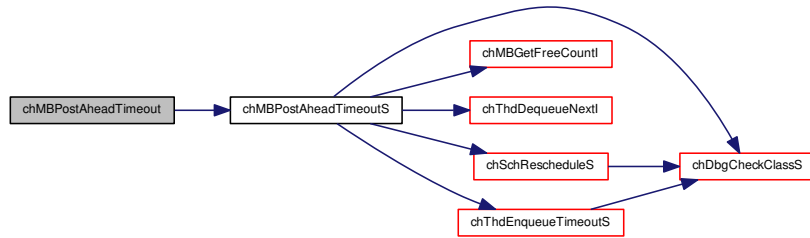
Return values

<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:



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

Returns

The operation status.

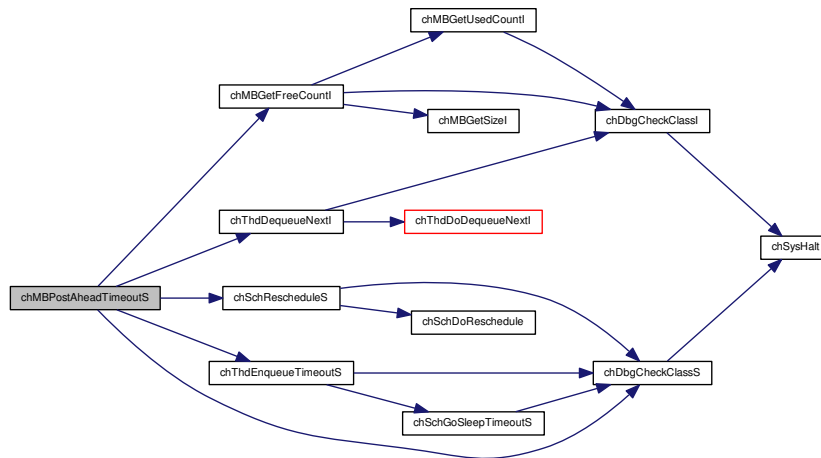
Return values

<code>MSG_OK</code>	if a message has been correctly posted.
<code>MSG_RESET</code>	if the mailbox has been reset.
<code>MSG_TIMEOUT</code>	if the operation has timed out.

Function Class:

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:



6.8.3.9 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.

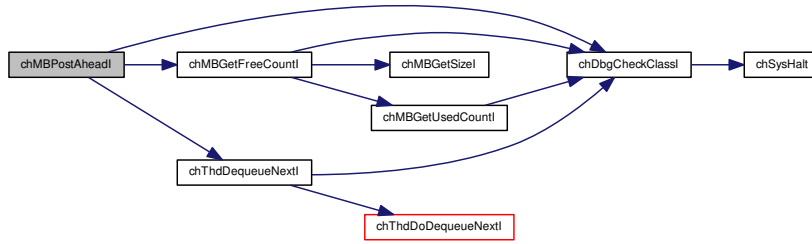
Return values

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



6.8.3.10 `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 <code>mailbox_t</code> object
out	<i>msgp</i>	pointer to a message variable for the received message
in	<i>timeout</i>	the number of ticks before the operation timeouts, the following special values are allowed: <ul style="list-style-type: none"> • <code>TIME_IMMEDIATE</code> immediate timeout. • <code>TIME_INFINITE</code> no timeout.

Returns

The operation status.

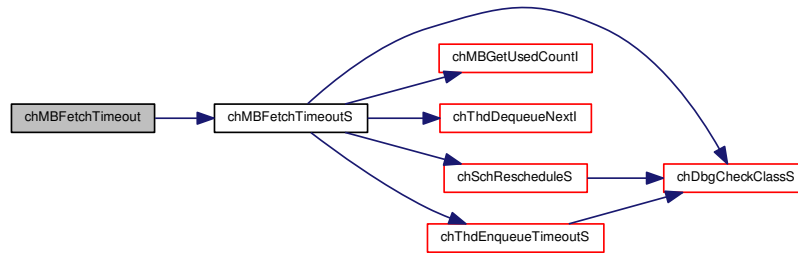
Return values

<code>MSG_OK</code>	if a message has been correctly fetched.
<code>MSG_RESET</code>	if the mailbox has been reset.
<code>MSG_TIMEOUT</code>	if the operation has timed out.

Function Class:

Normal API, 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 msg_t chMBFetchTimeoutS (mailbox_t * mbp, msg_t * msgp, sysinterval_t timeout)

Retrieves a message from a mailbox.

The invoking thread waits until a message is posted in the mailbox or the specified time runs out.

Parameters

in	<i>mbp</i>	the pointer to an initialized <code>mailbox_t</code> object
out	<i>msgp</i>	pointer to a message variable for the received message
in	<i>timeout</i>	the number of ticks before the operation timeouts, the following special values are allowed: <ul style="list-style-type: none"> • <code>TIME_IMMEDIATE</code> immediate timeout. • <code>TIME_INFINITE</code> no timeout.

Returns

The operation status.

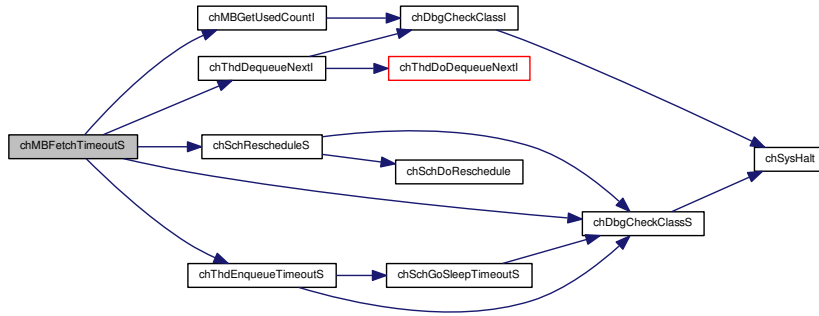
Return values

<code>MSG_OK</code>	if a message has been correctly fetched.
<code>MSG_RESET</code>	if the mailbox has been reset.
<code>MSG_TIMEOUT</code>	if the operation has timed out.

Function Class:

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

Here is the call graph for this function:



6.8.3.12 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 mailbox_t object
out	<i>msgp</i>	pointer to a message variable for the received message

Returns

The operation status.

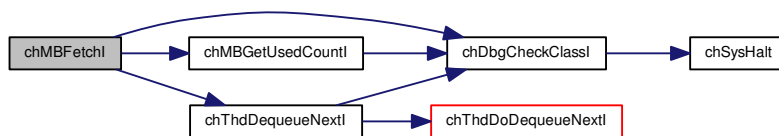
Return values

<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:



6.8.3.13 `static size_t chMBGetSizel (const mailbox_t * mbp) [inline],[static]`

Returns the mailbox buffer size as number of messages.

Parameters

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

Returns

The size of the mailbox.

Function Class:

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

6.8.3.14 `static size_t chMBGetUsedCountl (const mailbox_t * mbp) [inline],[static]`

Returns the number of used message slots into a mailbox.

Parameters

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

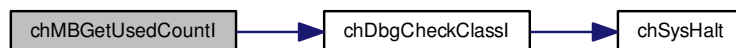
Returns

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:



6.8.3.15 `static size_t chMBGetFreeCountl (const mailbox_t * mbp) [inline],[static]`

Returns the number of free message slots into a mailbox.

Parameters

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

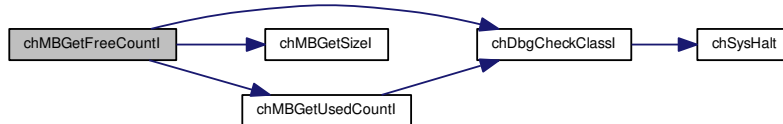
Returns

The number of empty message slots.

Function Class:

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

Here is the call graph for this function:



6.8.3.16 `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

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

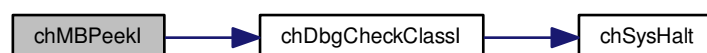
Returns

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:



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

Terminates the reset state.

Parameters

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

Function Class:

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

6.9 Pipes

6.9.1 Detailed Description

Macros

- #define `_PIPE_DATA`(name, buffer, size)
Data part of a static pipe initializer.
- #define `PIPE_DECL`(name, buffer, size) `pipe_t` name = `_PIPE_DATA`(name, buffer, size)
Static pipe initializer.

Data Structures

- struct `pipe_t`
Structure representing a pipe object.

Functions

- static `size_t` `pipe_write` (`pipe_t` *pp, const `uint8_t` *bp, `size_t` n)
Non-blocking pipe write.
- static `size_t` `pipe_read` (`pipe_t` *pp, `uint8_t` *bp, `size_t` n)
Non-blocking pipe read.
- void `chPipeObjectInit` (`pipe_t` *pp, `uint8_t` *buf, `size_t` n)
Initializes a `mailbox_t` object.
- void `chPipeReset` (`pipe_t` *pp)
Resets a `pipe_t` object.
- `size_t` `chPipeWriteTimeout` (`pipe_t` *pp, const `uint8_t` *bp, `size_t` n, `sysinterval_t` timeout)
Pipe write with timeout.
- `size_t` `chPipeReadTimeout` (`pipe_t` *pp, `uint8_t` *bp, `size_t` n, `sysinterval_t` timeout)
Pipe read with timeout.
- static `size_t` `chPipeGetSize` (const `pipe_t` *pp)
Returns the pipe buffer size as number of bytes.
- static `size_t` `chPipeGetUsedCount` (const `pipe_t` *pp)
Returns the number of used byte slots into a pipe.
- static `size_t` `chPipeGetFreeCount` (const `pipe_t` *pp)
Returns the number of free byte slots into a pipe.
- static void `chPipeResume` (`pipe_t` *pp)
Terminates the reset state.

6.9.2 Macro Definition Documentation

6.9.2.1 #define `_PIPE_DATA`(name, buffer, size)

Value:

```
{
    (uint8_t *) (buffer),
    (uint8_t *) (buffer) + size,
    (uint8_t *) (buffer),
    (uint8_t *) (buffer),
    (size_t) 0,
    false,
    NULL,
    NULL,
}
```

```

    _MUTEX_DATA (name.cmtx) ,
    _MUTEX_DATA (name.wmtx) ,
    _MUTEX_DATA (name.rmtx) ,
}

```

Data part of a static pipe initializer.

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

Parameters

in	<i>name</i>	the name of the pipe variable
in	<i>buffer</i>	pointer to the pipe buffer array of <code>uint8_t</code>
in	<i>size</i>	number of <code>uint8_t</code> elements in the buffer array

6.9.2.2 `#define PIPE_DECL(name, buffer, size) pipe_t name = _PIPE_DATA(name, buffer, size)`

Static pipe initializer.

Statically initialized pipes require no explicit initialization using `chPipeObjectInit()`.

Parameters

in	<i>name</i>	the name of the pipe variable
in	<i>buffer</i>	pointer to the pipe buffer array of <code>uint8_t</code>
in	<i>size</i>	number of <code>uint8_t</code> elements in the buffer array

6.9.3 Function Documentation

6.9.3.1 `static size_t pipe_write (pipe_t * pp, const uint8_t * bp, size_t n) [static]`

Non-blocking pipe write.

The function writes data from a buffer to a pipe. The operation completes when the specified amount of data has been transferred or when the pipe buffer has been filled.

Parameters

in	<i>pp</i>	the pointer to an initialized <code>pipe_t</code> object
in	<i>bp</i>	pointer to the data buffer
in	<i>n</i>	the maximum amount of data to be transferred, the value 0 is reserved

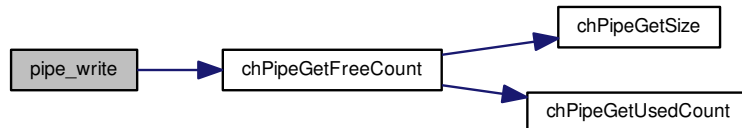
Returns

The number of bytes effectively transferred.

Function Class:

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

Here is the call graph for this function:



6.9.3.2 static size_t pipe_read (pipe_t * pp, uint8_t * bp, size_t n) [static]

Non-blocking pipe read.

The function reads data from a pipe into a buffer. The operation completes when the specified amount of data has been transferred or when the pipe buffer has been emptied.

Parameters

in	<i>pp</i>	the pointer to an initialized <code>pipe_t</code> object
out	<i>bp</i>	pointer to the data buffer
in	<i>n</i>	the maximum amount of data to be transferred, the value 0 is reserved

Returns

The number of bytes effectively transferred.

Function Class:

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

Here is the call graph for this function:



6.9.3.3 void chPipeObjectInit (pipe_t * pp, uint8_t * buf, size_t n)

Initializes a `mailbox_t` object.

Parameters

out	<i>pp</i>	the pointer to the <code>pipe_t</code> structure to be initialized
in	<i>buf</i>	pointer to the pipe buffer as an array of <code>uint8_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.

6.9.3.4 void chPipeReset (pipe_t * pp)

Resets a `pipe_t` object.

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

Postcondition

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

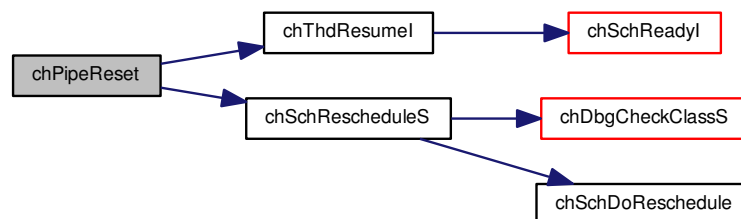
Parameters

in	<i>pp</i>	the pointer to an initialized <code>pipe_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:



6.9.3.5 size_t chPipeWriteTimeout (pipe_t * pp, const uint8_t * bp, size_t n, sysinterval_t timeout)

Pipe write with timeout.

The function writes data from a buffer to a pipe. The operation completes when the specified amount of data has been transferred or after the specified timeout or if the pipe has been reset.

Parameters

in	<i>pp</i>	the pointer to an initialized <code>pipe_t</code> object
in	<i>bp</i>	pointer to the data buffer
in	<i>n</i>	the number of bytes to be written, the value 0 is reserved
in	<i>timeout</i>	the number of ticks before the operation timeouts, the following special values are allowed: <ul style="list-style-type: none"> • <code>TIME_IMMEDIATE</code> immediate timeout. • <code>TIME_INFINITE</code> no timeout.

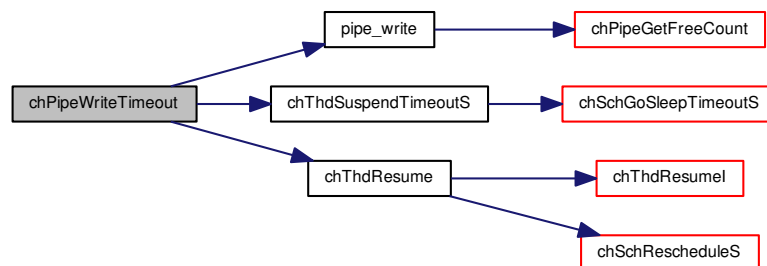
Returns

The number of bytes effectively transferred. A number lower than `n` means that a timeout occurred or the pipe went in reset state.

Function Class:

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

Here is the call graph for this function:



6.9.3.6 `size_t chPipeReadTimeout (pipe_t * pp, uint8_t * bp, size_t n, sysinterval_t timeout)`

Pipe read with timeout.

The function reads data from a pipe into a buffer. The operation completes when the specified amount of data has been transferred or after the specified timeout or if the pipe has been reset.

Parameters

in	<i>pp</i>	the pointer to an initialized <code>pipe_t</code> object
out	<i>bp</i>	pointer to the data buffer
in	<i>n</i>	the number of bytes to be read, the value 0 is reserved
in	<i>timeout</i>	the number of ticks before the operation timeouts, the following special values are allowed: <ul style="list-style-type: none"> • <code>TIME_IMMEDIATE</code> immediate timeout. • <code>TIME_INFINITE</code> no timeout.

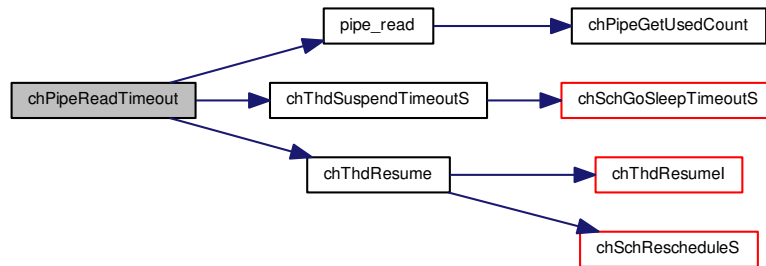
Returns

The number of bytes effectively transferred. A number lower than `n` means that a timeout occurred or the pipe went in reset state.

Function Class:

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

Here is the call graph for this function:



6.9.3.7 `static size_t chPipeGetSize (const pipe_t * pp) [inline],[static]`

Returns the pipe buffer size as number of bytes.

Parameters

<code>in</code>	<code>pp</code>	the pointer to an initialized <code>pipe_t</code> object
-----------------	-----------------	--

Returns

The size of the pipe.

Function Class:

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

6.9.3.8 `static size_t chPipeGetUsedCount (const pipe_t * pp) [inline],[static]`

Returns the number of used byte slots into a pipe.

Parameters

<code>in</code>	<code>pp</code>	the pointer to an initialized <code>pipe_t</code> object
-----------------	-----------------	--

Returns

The number of queued bytes.

Function Class:

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

6.9.3.9 `static size_t chPipeGetFreeCount (const pipe_t * pp) [inline],[static]`

Returns the number of free byte slots into a pipe.

Parameters

in	<i>pp</i>	the pointer to an initialized <code>pipe_t</code> object
----	-----------	--

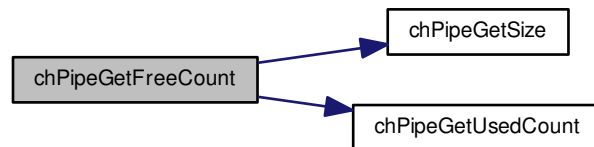
Returns

The number of empty byte slots.

Function Class:

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

Here is the call graph for this function:



6.9.3.10 `static void chPipeResume (pipe_t * pp) [inline],[static]`

Terminates the reset state.

Parameters

in	<i>pp</i>	the pointer to an initialized <code>pipe_t</code> object
----	-----------	--

Function Class:

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

6.10 Memory Management

6.10.1 Detailed Description

Memory Management services.

Modules

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

6.11 Core Memory Manager

6.11.1 Detailed Description

Core Memory Manager related APIs and services.

Operation mode

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

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

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

By having a centralized memory provider the various allocators can coexist and share the main memory. This allocator, alone, is also useful for very simple applications that just require a simple way to get memory blocks.

Precondition

In order to use the core memory manager APIs the `CH_CFG_USE_MEMCORE` option must be enabled in [chconf.h](#).

Note

Compatible with RT and NIL.

Macros

- `#define CH_CFG_MEMCORE_SIZE 0`
Managed RAM size.

Typedefs

- `typedef void *(* memgetfunc_t) (size_t size, unsigned align)`
Memory get function.
- `typedef void *(* memgetfunc2_t) (size_t size, unsigned align, size_t offset)`
Enhanced memory get function.

Data Structures

- `struct memcore_t`
Type of memory core object.

Functions

- `void _core_init (void)`
Low level memory manager initialization.
- `void * 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.

6.11.2 Macro Definition Documentation

6.11.2.1 #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`.

6.11.3 Typedef Documentation

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

Memory get function.

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

Enhanced memory get function.

6.11.4 Function Documentation

6.11.4.1 void_core_init (void)

Low level memory manager initialization.

Function Class:

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

6.11.4.2 `void * chCoreAllocAlignedWithOffset1 (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:



6.11.4.3 `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.

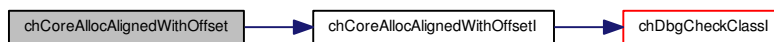
Return values

<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:

6.11.4.4 `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.

6.11.4.5 `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.

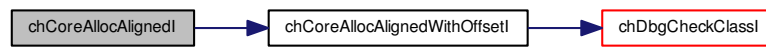
Return values

<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:



6.11.4.6 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.

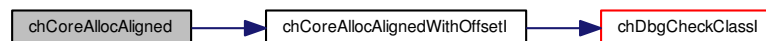
Return values

<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:



6.11.4.7 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

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

Returns

A pointer to the allocated memory block.

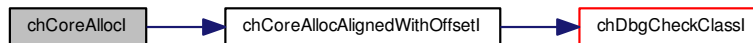
Return values

<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:

**6.11.4.8 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.

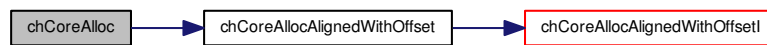
Return values

<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:



6.11.5 Variable Documentation

6.11.5.1 `memcore_t ch_memcore`

Memory core descriptor.

6.12 Memory Heaps

6.12.1 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.

6.12.2 Macro Definition Documentation

6.12.2.1 #define CH_HEAP_ALIGNMENT 8U

Minimum alignment used for heap.

Note

Cannot use the sizeof operator in this macro.

6.12.2.2 #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.

6.12.3 Typedef Documentation

6.12.3.1 typedef struct memory_heap memory_heap_t

Type of a memory heap.

6.12.3.2 typedef union heap_header heap_header_t

Type of a memory heap header.

6.12.4 Function Documentation

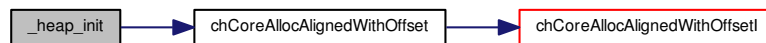
6.12.4.1 void_heap_init (void)

Initializes the default heap.

Function Class:

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

Here is the call graph for this function:



6.12.4.2 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.

6.12.4.3 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 <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.
in	<i>align</i>	desired memory alignment

Returns

A pointer to the aligned allocated block.

Return values

<code>NULL</code>	if the block cannot be allocated.
-------------------	-----------------------------------

Function Class:

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

6.12.4.4 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.

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

Returns

The number of fragments in the heap.

Function Class:

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

6.12.4.6 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 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.

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:



6.12.4.7 `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

<code>in</code>	<code>p</code>	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.

6.12.5 Variable Documentation

6.12.5.1 `memory_heap_t default_heap [static]`

Default heap descriptor.

6.13 Memory Pools

6.13.1 Detailed Description

Memory Pools related APIs and services.

Operation mode

The Memory Pools APIs allow to allocate/free fixed size objects in **constant time** and reliably without memory fragmentation problems.

Memory Pools do not enforce any alignment constraint on the contained object however the objects must be properly aligned to contain a pointer to void.

Precondition

In order to use the memory pools APIs the `CH_CFG_USE_MEMPOOLS` option must be enabled in `chconf.h`.

Note

Compatible with RT and NIL.

Macros

- `#define _MEMORYPOOL_DATA(name, size, align, provider) {NULL, size, align, provider}`
Data part of a static memory pool initializer.
- `#define MEMORYPOOL_DECL(name, size, align, provider) memory_pool_t name = _MEMORYPOOL_DATA(name, size, align, provider)`
Static memory pool initializer.
- `#define _GUARDEDMEMORYPOOL_DATA(name, size, align)`
Data part of a static guarded memory pool initializer.
- `#define GUARDEDMEMORYPOOL_DECL(name, size, align) guarded_memory_pool_t name = _GUARDEDMEMORYPOOL_DATA(name, size, align)`
Static guarded memory pool initializer.

Data Structures

- struct `pool_header`
Memory pool free object header.
- struct `memory_pool_t`
Memory pool descriptor.
- struct `guarded_memory_pool_t`
Guarded memory pool descriptor.

Functions

- void `chPoolObjectInitAligned (memory_pool_t *mp, size_t size, unsigned align, memgetfunc_t provider)`
Initializes an empty memory pool.
- void `chPoolLoadArray (memory_pool_t *mp, void *p, size_t n)`
Loads a memory pool with an array of static objects.
- void * `chPoolAlloc (memory_pool_t *mp)`
Allocates an object from a memory pool.

- void * `chPoolAlloc` (`memory_pool_t *mp`)
Allocates an object from a memory pool.
- void `chPoolFree` (`memory_pool_t *mp`, void *objp)
Releases an object into a memory pool.
- void `chPoolFree` (`memory_pool_t *mp`, void *objp)
Releases an object into a memory pool.
- void `chGuardedPoolObjectInitAligned` (`guarded_memory_pool_t *gmp`, size_t size, unsigned align)
Initializes an empty guarded memory pool.
- void `chGuardedPoolLoadArray` (`guarded_memory_pool_t *gmp`, void *p, size_t n)
Loads a guarded memory pool with an array of static objects.
- void * `chGuardedPoolAllocTimeoutS` (`guarded_memory_pool_t *gmp`, `sysinterval_t` timeout)
Allocates an object from a guarded memory pool.
- void * `chGuardedPoolAllocTimeout` (`guarded_memory_pool_t *gmp`, `sysinterval_t` timeout)
Allocates an object from a guarded memory pool.
- void `chGuardedPoolFree` (`guarded_memory_pool_t *gmp`, void *objp)
Releases an object into a guarded memory pool.
- static void `chPoolObjectInit` (`memory_pool_t *mp`, size_t size, `memgetfunc_t` provider)
Initializes an empty memory pool.
- static void `chPoolAdd` (`memory_pool_t *mp`, void *objp)
Adds an object to a memory pool.
- static void `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 cnt_t `chGuardedPoolGetCounterI` (`guarded_memory_pool_t *gmp`)
Gets the count of objects in a guarded memory pool.
- static void * `chGuardedPoolAllocI` (`guarded_memory_pool_t *gmp`)
Allocates an object from a guarded memory pool.
- static void `chGuardedPoolFreeI` (`guarded_memory_pool_t *gmp`, void *objp)
Releases an object into a guarded memory pool.
- static void `chGuardedPoolFreeS` (`guarded_memory_pool_t *gmp`, void *objp)
Releases an object into a guarded memory pool.
- static void `chGuardedPoolAdd` (`guarded_memory_pool_t *gmp`, void *objp)
Adds an object to a guarded memory pool.
- static void `chGuardedPoolAddI` (`guarded_memory_pool_t *gmp`, void *objp)
Adds an object to a guarded memory pool.
- static void `chGuardedPoolAddS` (`guarded_memory_pool_t *gmp`, void *objp)
Adds an object to a guarded memory pool.

6.13.2 Macro Definition Documentation

6.13.2.1 #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

6.13.2.2 `#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 NULL if the pool is not allowed to grow automatically

6.13.2.3 `#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

6.13.2.4 `#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

6.13.3 Function Documentation

6.13.3.1 `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.

6.13.3.2 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.

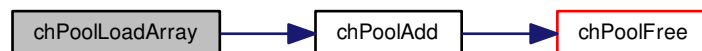
Parameters

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:



6.13.3.3 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:

**6.13.3.4 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.

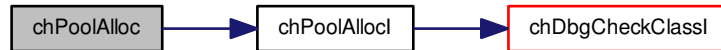
Return values

<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:



6.13.3.5 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.

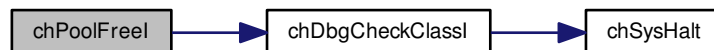
Parameters

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:



6.13.3.6 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.

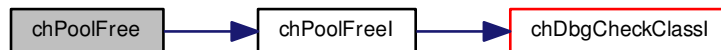
Parameters

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:



6.13.3.7 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:



6.13.3.8 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.

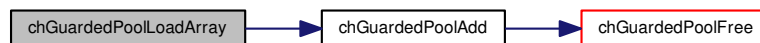
Parameters

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:



6.13.3.9 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.

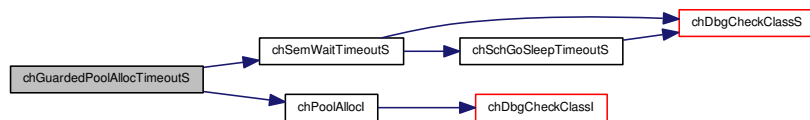
Return values

<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:



6.13.3.10 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.

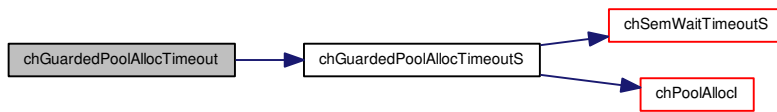
Return values

<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:



6.13.3.11 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.

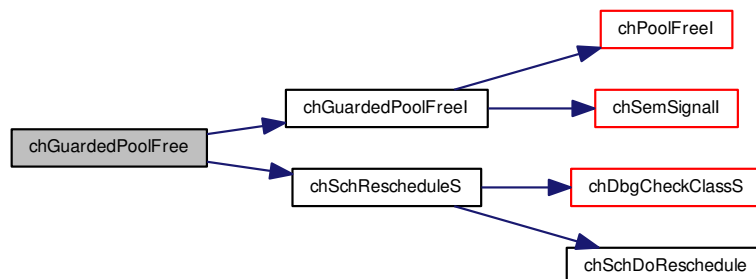
Parameters

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:

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

Here is the call graph for this function:



6.13.3.12 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:



6.13.3.13 static void chPoolAdd ([memory_pool_t](#) * *mp*, void * *objp*) [inline], [static]

Adds an object to a memory pool.

Precondition

- The memory pool must be already been initialized.
- The added object must be of the right size for the specified memory pool.
- The added object must be properly aligned.

Note

This function is just an alias for [chPoolFree\(\)](#) and has been added for clarity.

Parameters

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

Function Class:

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

Here is the call graph for this function:



6.13.3.14 `static void chPoolAddI (memory_pool_t * mp, void * objp)` [inline],[static]

Adds an object to a memory pool.

Precondition

The memory pool must be already been initialized.
 The added object must be of the right size for the specified memory pool.
 The added object must be properly aligned.

Note

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

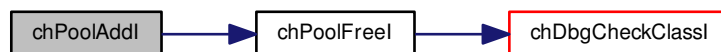
Parameters

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

Function Class:

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

Here is the call graph for this function:



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

Initializes an empty guarded memory pool.

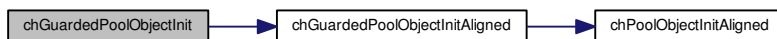
Parameters

out	<i>gmp</i>	pointer to a 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.

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:



6.13.3.16 static cnt_t chGuardedPoolGetCounterl (guarded_memory_pool_t* *gmp*) [inline],[static]

Gets the count of objects in a guarded memory pool.

Precondition

The guarded memory pool must be already been initialized.

Parameters

in	<i>gmp</i>	pointer to a guarded_memory_pool_t structure
----	------------	--

Returns

The number of objects.

Function Class:

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

6.13.3.17 static void* chGuardedPoolAlloc1 (guarded_memory_pool_t* *gmp*) [inline],[static]

Allocates an object from a guarded memory pool.

Precondition

The guarded memory pool must be already been initialized.

Parameters

in	<i>gmp</i>	pointer to a guarded_memory_pool_t structure
----	------------	--

Returns

The pointer to the allocated object.

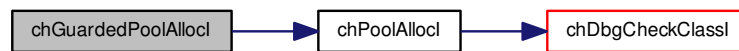
Return values

<i>NULL</i>	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:



6.13.3.18 `static void chGuardedPoolFree (guarded_memory_pool_t * gmp, void * objp) [inline], [static]`

Releases an object into a guarded memory pool.

Precondition

The guarded memory pool must already be initialized.
 The freed object must be of the right size for the specified guarded memory pool.
 The added object must be properly aligned.

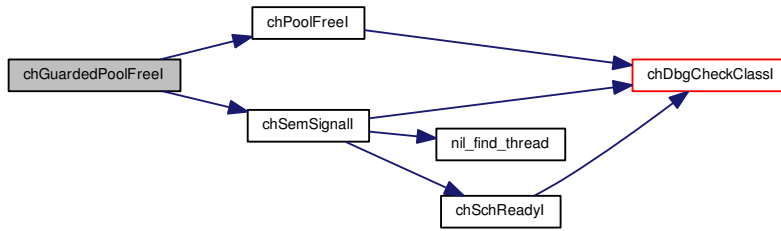
Parameters

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

Function Class:

This is an **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:



6.13.3.19 static void chGuardedPoolFreeS (guarded_memory_pool_t * gmp, void * objp) [inline],[static]

Releases an object into a guarded memory pool.

Precondition

- The guarded memory pool must already be initialized.
- The freed object must be of the right size for the specified guarded memory pool.
- The added object must be properly aligned.

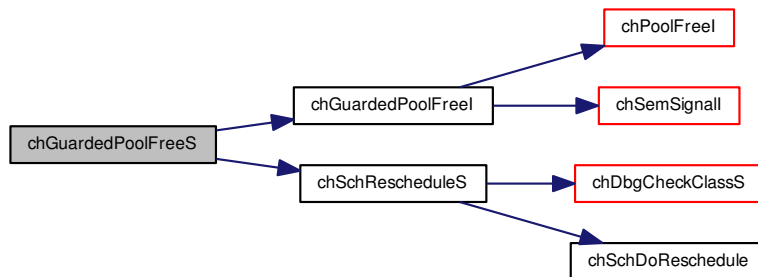
Parameters

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

Function Class:

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

Here is the call graph for this function:



6.13.3.20 `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.

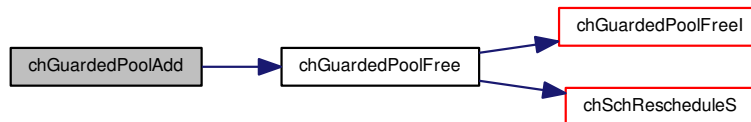
Parameters

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:



6.13.3.21 `static void chGuardedPoolAddI (guarded_memory_pool_t * gmp, void * objp) [inline],[static]`

Adds an object to a guarded memory pool.

Precondition

The guarded memory pool must be already been initialized.
 The added object must be of the right aligned 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.

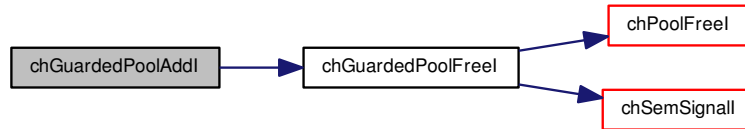
Parameters

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:



6.13.3.22 `static void chGuardedPoolAddS (guarded_memory_pool_t * gmp, void * objp) [inline], [static]`

Adds an object to a guarded memory pool.

Precondition

The guarded memory pool must be already been initialized.
 The added object must be of the right size for the specified guarded memory pool.
 The added object must be properly aligned.

Note

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

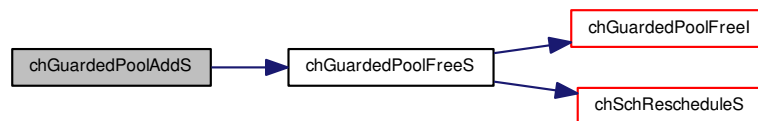
Parameters

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

Function Class:

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

Here is the call graph for this function:



6.14 Complex Services

6.14.1 Detailed Description

Modules

- [Objects FIFOs](#)
- [Dynamic Objects Factory](#)

6.15 Objects FIFOs

6.15.1 Detailed Description

Typedefs

- typedef struct [ch_objects_fifo](#) [objects_fifo_t](#)
Type of an objects FIFO.

Data Structures

- struct [ch_objects_fifo](#)
Type of an objects FIFO.

Functions

- static void [chFifoObjectInitAligned](#) ([objects_fifo_t](#) *ofp, size_t objsize, size_t objn, unsigned objalign, void *objbuf, msg_t *msgbuf)
Initializes a FIFO object.
- static void [chFifoObjectInit](#) ([objects_fifo_t](#) *ofp, size_t objsize, size_t objn, void *objbuf, msg_t *msgbuf)
Initializes a FIFO object.
- static void * [chFifoTakeObjectI](#) ([objects_fifo_t](#) *ofp)
Allocates a free object.
- static void * [chFifoTakeObjectTimeoutS](#) ([objects_fifo_t](#) *ofp, [sysinterval_t](#) timeout)
Allocates a free object.
- static void * [chFifoTakeObjectTimeout](#) ([objects_fifo_t](#) *ofp, [sysinterval_t](#) timeout)
Allocates a free object.
- static void [chFifoReturnObjectI](#) ([objects_fifo_t](#) *ofp, void *objp)
Releases a fetched object.
- static void [chFifoReturnObjectS](#) ([objects_fifo_t](#) *ofp, void *objp)
Releases a fetched object.
- static void [chFifoReturnObject](#) ([objects_fifo_t](#) *ofp, void *objp)
Releases a fetched object.
- static void [chFifoSendObjectI](#) ([objects_fifo_t](#) *ofp, void *objp)
Posts an object.
- static void [chFifoSendObjectS](#) ([objects_fifo_t](#) *ofp, void *objp)
Posts an object.
- static void [chFifoSendObject](#) ([objects_fifo_t](#) *ofp, void *objp)
Posts an object.
- static void [chFifoSendObjectAheadI](#) ([objects_fifo_t](#) *ofp, void *objp)
Posts an high priority object.
- static void [chFifoSendObjectAheadS](#) ([objects_fifo_t](#) *ofp, void *objp)
Posts an high priority object.
- static void [chFifoSendObjectAhead](#) ([objects_fifo_t](#) *ofp, void *objp)
Posts an high priority object.
- static msg_t [chFifoReceiveObjectI](#) ([objects_fifo_t](#) *ofp, void **objpp)
Fetches an object.
- static msg_t [chFifoReceiveObjectTimeoutS](#) ([objects_fifo_t](#) *ofp, void **objpp, [sysinterval_t](#) timeout)
Fetches an object.
- static msg_t [chFifoReceiveObjectTimeout](#) ([objects_fifo_t](#) *ofp, void **objpp, [sysinterval_t](#) timeout)
Fetches an object.

6.15.2 Typedef Documentation

6.15.2.1 typedef struct ch_objects_fifo objects_fifo_t

Type of an objects FIFO.

6.15.3 Function Documentation

6.15.3.1 static void chFifoObjectInitAligned (objects_fifo_t * ofp, size_t objsize, size_t objn, unsigned objalign, void * objbuf, msg_t * msgbuf) [inline],[static]

Initializes a FIFO object.

Precondition

The messages size must be a multiple of the alignment requirement.

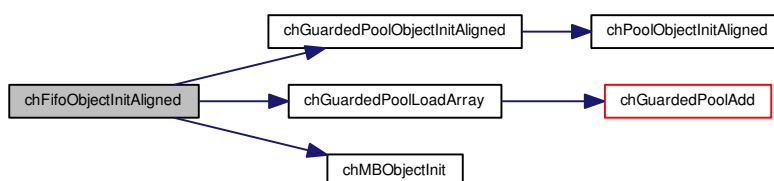
Parameters

out	<i>ofp</i>	pointer to a <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:



6.15.3.2 static void chFifoObjectInit (objects_fifo_t * ofp, size_t objsize, size_t objn, void * objbuf, msg_t * msgbuf) [inline],[static]

Initializes a FIFO object.

Precondition

The messages size must be a multiple of the alignment requirement.

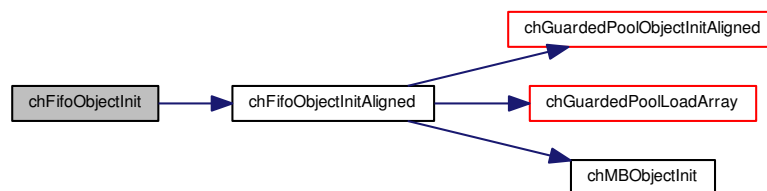
Parameters

out	<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>objbuf</i>	pointer to the buffer of objects, it must be able to hold <i>objn</i> objects of <i>objsize</i> size with <i>objealign</i> alignment
in	<i>msgbuf</i>	pointer to the buffer of messages, it must be able to hold <i>objn</i> messages

Function Class:

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

Here is the call graph for this function:



6.15.3.3 `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.

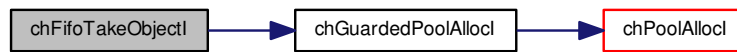
Return values

<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:



6.15.3.4 `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.

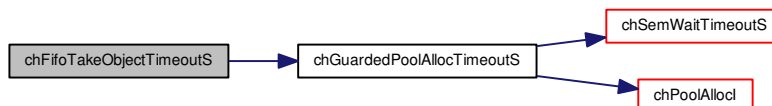
Return values

<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:



6.15.3.5 `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.

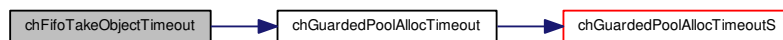
Return values

<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:

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

Releases a fetched object.

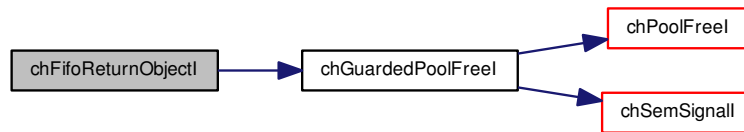
Parameters

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:

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:



6.15.3.7 `static void chFifoReturnObjectS (objects_fifo_t * ofp, void * objp) [inline],[static]`

Releases a fetched object.

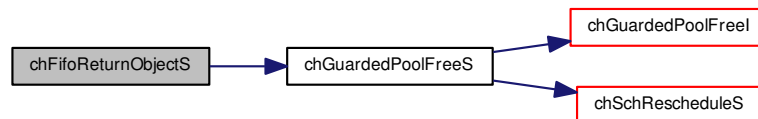
Parameters

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 **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:



6.15.3.8 `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:



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

Posts an object.

Note

By design the object can be always immediately posted.

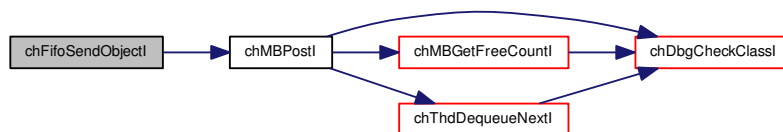
Parameters

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:



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

Posts an object.

Note

By design the object can be always immediately posted.

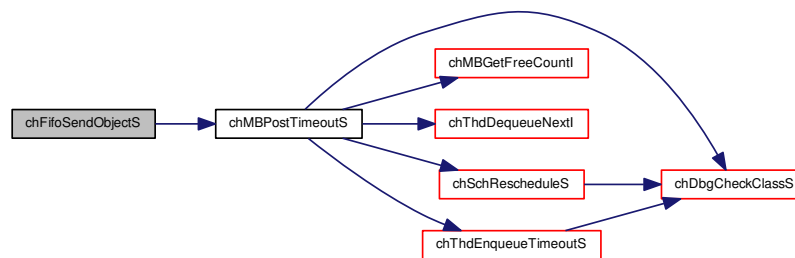
Parameters

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:



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

Posts an object.

Note

By design the object can be always immediately posted.

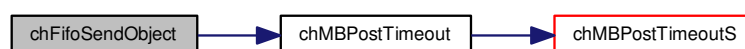
Parameters

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:



6.15.3.12 `static void chFifoSendObjectAheadI (objects_fifo_t * ofp, void * objp) [inline],[static]`

Posts an high priority object.

Note

By design the object can be always immediately posted.

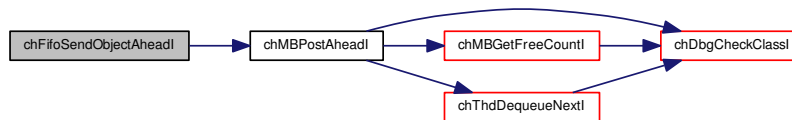
Parameters

in	<i>ofp</i>	pointer to a <code>objects_fifo_t</code> structure
in	<i>objp</i>	pointer to the object to be posted

Function Class:

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

Here is the call graph for this function:



6.15.3.13 `static void chFifoSendObjectAheadS (objects_fifo_t * ofp, void * objp) [inline],[static]`

Posts an high priority object.

Note

By design the object can be always immediately posted.

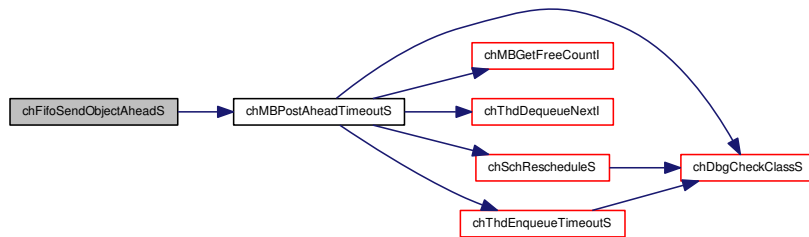
Parameters

in	<i>ofp</i>	pointer to a <code>objects_fifo_t</code> structure
in	<i>objp</i>	pointer to the object to be posted

Function Class:

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

Here is the call graph for this function:



6.15.3.14 `static void chFifoSendObjectAhead (objects_fifo_t * ofp, void * objp) [inline], [static]`

Posts an high priority object.

Note

By design the object can be always immediately posted.

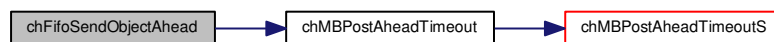
Parameters

in	<i>ofp</i>	pointer to a <code>objects_fifo_t</code> structure
in	<i>objp</i>	pointer to the object to be released

Function Class:

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

Here is the call graph for this function:



6.15.3.15 `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.

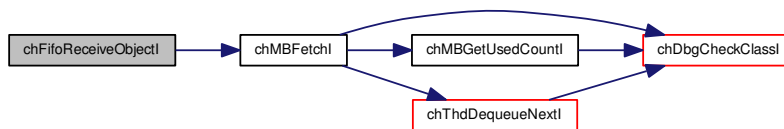
Return values

<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:



6.15.3.16 `static msg_t chFifoReceiveObjectTimeoutS (objects_fifo_t * ofp, void ** objpp, sysinterval_t timeout)`
`[inline],[static]`

Fetches an object.

Parameters

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

Returns

The operation status.

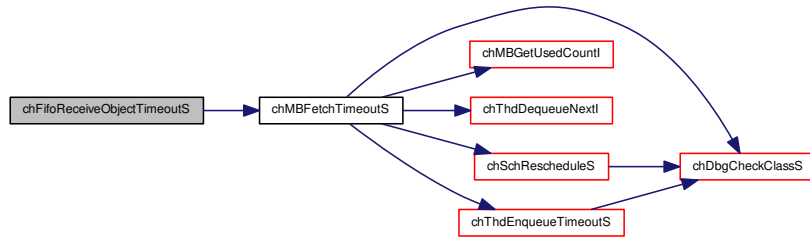
Return values

<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:



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

Fetches an object.

Parameters

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

Returns

The operation status.

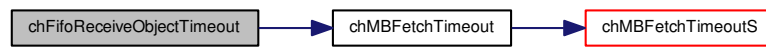
Return values

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

Function Class:

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

Here is the call graph for this function:



6.16 Dynamic Objects Factory

6.16.1 Detailed Description

The object factory is a subsystem that allows to:

- Register static objects by name.
- Dynamically create objects and assign them a name.
- Retrieve existing objects by name.
- Free objects by reference.

Allocated OS objects are handled using a reference counter, only when all references have been released then the object memory is freed in a pool.

Precondition

This subsystem requires the `CH_CFG_USE_MEMCORE` and `CH_CFG_USE_MEMPOOLS` options to be set to `TRUE`. The option `CH_CFG_USE_HEAP` is also required if the support for variable length objects is enabled.

Note

Compatible with RT and NIL.

Macros

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

Typedefs

- typedef struct [ch_dyn_element](#) [dyn_element_t](#)
Type of a dynamic object list element.
- typedef struct [ch_dyn_list](#) [dyn_list_t](#)
Type of a dynamic object list.
- typedef struct [ch_registered_static_object](#) [registered_object_t](#)
Type of a registered object.
- typedef struct [ch_dyn_object](#) [dyn_buffer_t](#)
Type of a dynamic buffer object.
- typedef struct [ch_dyn_semaphore](#) [dyn_semaphore_t](#)
Type of a dynamic semaphore.
- typedef struct [ch_dyn_mailbox](#) [dyn_mailbox_t](#)
Type of a dynamic buffer object.
- typedef struct [ch_dyn_objects_fifo](#) [dyn_objects_fifo_t](#)
Type of a dynamic buffer object.
- typedef struct [ch_dyn_pipe](#) [dyn_pipe_t](#)
Type of a dynamic pipe object.
- typedef struct [ch_objects_factory](#) [objects_factory_t](#)
Type of the factory main object.

Data Structures

- struct [ch_dyn_element](#)
Type of a dynamic object list element.
- struct [ch_dyn_list](#)
Type of a dynamic object list.
- struct [ch_registered_static_object](#)
Type of a registered object.
- struct [ch_dyn_object](#)
Type of a dynamic buffer object.
- struct [ch_dyn_semaphore](#)
Type of a dynamic semaphore.
- struct [ch_dyn_mailbox](#)
Type of a dynamic buffer object.
- struct [ch_dyn_objects_fifo](#)
Type of a dynamic buffer object.
- struct [ch_dyn_pipe](#)
Type of a dynamic pipe object.
- struct [ch_objects_factory](#)
Type of the factory main object.

Functions

- void [_factory_init](#) (void)
Initializes the objects factory.
- [registered_object_t](#) * [chFactoryRegisterObject](#) (const char *name, void *objp)
Registers a generic object.
- [registered_object_t](#) * [chFactoryFindObject](#) (const char *name)
Retrieves a registered object.

- [registered_object_t * chFactoryFindObjectByPointer](#) (void *objp)
Retrieves a registered object by pointer.
- void [chFactoryReleaseObject](#) (registered_object_t *rop)
Releases a registered object.
- [dyn_buffer_t * chFactoryCreateBuffer](#) (const char *name, size_t size)
Creates a generic dynamic buffer object.
- [dyn_buffer_t * chFactoryFindBuffer](#) (const char *name)
Retrieves a dynamic buffer object.
- void [chFactoryReleaseBuffer](#) (dyn_buffer_t *dbp)
Releases a dynamic buffer object.
- [dyn_semaphore_t * chFactoryCreateSemaphore](#) (const char *name, cnt_t n)
Creates a dynamic semaphore object.
- [dyn_semaphore_t * chFactoryFindSemaphore](#) (const char *name)
Retrieves a dynamic semaphore object.
- void [chFactoryReleaseSemaphore](#) (dyn_semaphore_t *dsp)
Releases a dynamic semaphore object.
- [dyn_mailbox_t * chFactoryCreateMailbox](#) (const char *name, size_t n)
Creates a dynamic mailbox object.
- [dyn_mailbox_t * chFactoryFindMailbox](#) (const char *name)
Retrieves a dynamic mailbox object.
- void [chFactoryReleaseMailbox](#) (dyn_mailbox_t *dmp)
Releases a dynamic mailbox object.
- [dyn_objects_fifo_t * chFactoryCreateObjectsFIFO](#) (const char *name, size_t objsize, size_t objn, unsigned objalign)
Creates a dynamic "objects FIFO" object.
- [dyn_objects_fifo_t * chFactoryFindObjectsFIFO](#) (const char *name)
Retrieves a dynamic "objects FIFO" object.
- void [chFactoryReleaseObjectsFIFO](#) (dyn_objects_fifo_t *dofp)
Releases a dynamic "objects FIFO" object.
- [dyn_pipe_t * chFactoryCreatePipe](#) (const char *name, size_t size)
Creates a dynamic pipe object.
- [dyn_pipe_t * chFactoryFindPipe](#) (const char *name)
Retrieves a dynamic pipe object.
- void [chFactoryReleasePipe](#) (dyn_pipe_t *dpp)
Releases a dynamic pipe object.
- static [dyn_element_t * chFactoryDuplicateReference](#) (dyn_element_t *dep)
Duplicates an object reference.
- static void * [chFactoryGetObject](#) (registered_object_t *rop)
Returns the pointer to the inner registered object.
- static size_t [chFactoryGetBufferSize](#) (dyn_buffer_t *dbp)
Returns the size of a generic dynamic buffer object.
- static uint8_t * [chFactoryGetBuffer](#) (dyn_buffer_t *dbp)
Returns the pointer to the inner buffer.
- static [semaphore_t * chFactoryGetSemaphore](#) (dyn_semaphore_t *dsp)
Returns the pointer to the inner semaphore.
- static [mailbox_t * chFactoryGetMailbox](#) (dyn_mailbox_t *dmp)
Returns the pointer to the inner mailbox.
- static [objects_fifo_t * chFactoryGetObjectsFIFO](#) (dyn_objects_fifo_t *dofp)
Returns the pointer to the inner objects FIFO.
- static [pipe_t * chFactoryGetPipe](#) (dyn_pipe_t *dpp)
Returns the pointer to the inner pipe.

Variables

- [objects_factory_t ch_factory](#)
Factory object static instance.

6.16.2 Macro Definition Documentation

6.16.2.1 #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.

6.16.2.2 #define CH_CFG_FACTORY_OBJECTS_REGISTRY TRUE

Enables the registry of generic objects.

6.16.2.3 #define CH_CFG_FACTORY_GENERIC_BUFFERS TRUE

Enables factory for generic buffers.

6.16.2.4 #define CH_CFG_FACTORY_SEMAPHORES TRUE

Enables factory for semaphores.

6.16.2.5 #define CH_CFG_FACTORY_SEMAPHORES FALSE

Enables factory for semaphores.

6.16.2.6 #define CH_CFG_FACTORY_MAILBOXES TRUE

Enables factory for mailboxes.

6.16.2.7 #define CH_CFG_FACTORY_MAILBOXES FALSE

Enables factory for mailboxes.

6.16.2.8 #define CH_CFG_FACTORY_OBJ_FIFOS TRUE

Enables factory for objects FIFOs.

6.16.2.9 #define CH_CFG_FACTORY_OBJ_FIFOS TRUE

Enables factory for objects FIFOs.

6.16.2.10 #define CH_CFG_FACTORY_OBJ_FIFOS FALSE

Enables factory for objects FIFOs.

6.16.2.11 `#define CH_CFG_FACTORY_PIPES TRUE`

Enables factory for Pipes.

6.16.2.12 `#define CH_CFG_FACTORY_PIPES FALSE`

Enables factory for Pipes.

6.16.3 Typedef Documentation

6.16.3.1 `typedef struct ch_dyn_element dyn_element_t`

Type of a dynamic object list element.

6.16.3.2 `typedef struct ch_dyn_list dyn_list_t`

Type of a dynamic object list.

6.16.3.3 `typedef struct ch_registered_static_object registered_object_t`

Type of a registered object.

6.16.3.4 `typedef struct ch_dyn_object dyn_buffer_t`

Type of a dynamic buffer object.

6.16.3.5 `typedef struct ch_dyn_semaphore dyn_semaphore_t`

Type of a dynamic semaphore.

6.16.3.6 `typedef struct ch_dyn_mailbox dyn_mailbox_t`

Type of a dynamic buffer object.

6.16.3.7 `typedef struct ch_dyn_objects_fifo dyn_objects_fifo_t`

Type of a dynamic buffer object.

6.16.3.8 `typedef struct ch_dyn_pipe dyn_pipe_t`

Type of a dynamic pipe object.

6.16.3.9 `typedef struct ch_objects_factory objects_factory_t`

Type of the factory main object.

6.16.4 Function Documentation

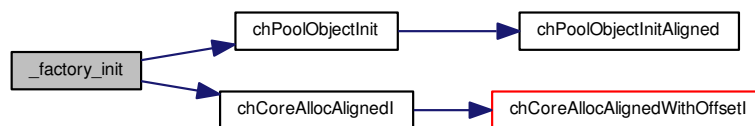
6.16.4.1 void _factory_init (void)

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:



6.16.4.2 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.

6.16.4.3 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.

6.16.4.4 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.

6.16.4.5 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.

6.16.4.6 `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.

6.16.4.7 `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	<i>name</i>	name of the dynamic buffer object
----	-------------	-----------------------------------

Returns

The reference to the found dynamic buffer object.

Return values

<i>NULL</i>	if a dynamic buffer object with the specified name does not exist.
-------------	--

Function Class:

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

6.16.4.8 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	<i>dbp</i>	dynamic buffer object reference
----	------------	---------------------------------

Function Class:

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

6.16.4.9 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.

6.16.4.10 `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

<i>in</i>	<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.

6.16.4.11 `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

<i>in</i>	<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.

6.16.4.12 `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.

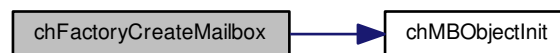
Return values

<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:

**6.16.4.13 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.

6.16.4.14 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.

6.16.4.15 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.

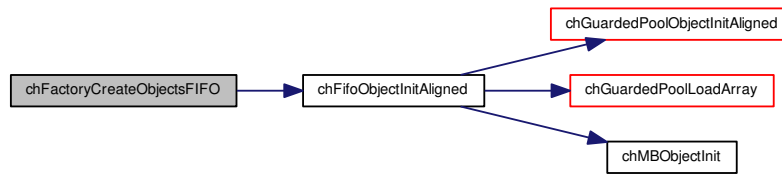
Return values

<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:



6.16.4.16 `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.

6.16.4.17 `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.

6.16.4.18 `dyn_pipe_t * chFactoryCreatePipe (const char * name, size_t size)`

Creates a dynamic pipe object.

Postcondition

A reference to the dynamic pipe object is returned and the reference counter is initialized to one.
The dynamic pipe object is initialized and ready to use.

Parameters

in	<i>name</i>	name to be assigned to the new dynamic pipe object
in	<i>size</i>	pipe buffer size

Returns

The reference to the created dynamic pipe object.

Return values

<i>NULL</i>	if the dynamic pipe object cannot be allocated or a dynamic pipe object with the same name exists.
-------------	--

Function Class:

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

Here is the call graph for this function:

6.16.4.19 `dyn_pipe_t * chFactoryFindPipe (const char * name)`

Retrieves a dynamic pipe object.

Postcondition

A reference to the dynamic pipe object is returned with the reference counter increased by one.

Parameters

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

Returns

The reference to the found dynamic pipe object.

Return values

<i>NULL</i>	if a dynamic pipe object with the specified name does not exist.
-------------	--

Function Class:

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

6.16.4.20 void chFactoryReleasePipe (dyn_pipe_t * dpp)

Releases a dynamic pipe object.

The reference counter of the dynamic pipe object is decreased by one, if reaches zero then the dynamic pipe object memory is freed.

Parameters

in	<i>dpp</i>	dynamic pipe object reference
----	------------	-------------------------------

Function Class:

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

6.16.4.21 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.

6.16.4.22 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.

6.16.4.23 `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
----	------------	---------------------------------

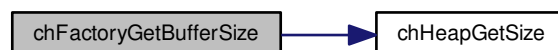
Returns

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:



6.16.4.24 `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.

6.16.4.25 `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.

6.16.4.26 `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.

6.16.4.27 `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.

6.16.4.28 `static pipe_t* chFactoryGetPipe (dyn_pipe_t* dpp) [inline],[static]`

Returns the pointer to the inner pipe.

Parameters

in	<i>dpp</i>	dynamic pipe object reference
----	------------	-------------------------------

Returns

The pointer to the pipe.

Function Class:

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

6.16.5 Variable Documentation

6.16.5.1 `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.

Chapter 7

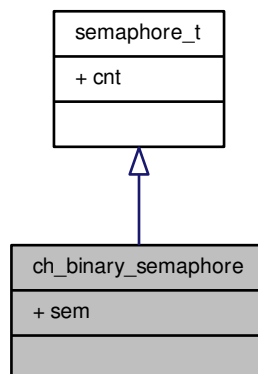
Data Structure Documentation

7.1 ch_binary_semaphore Struct Reference

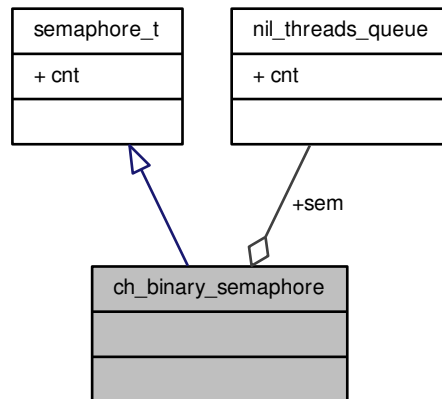
Binary semaphore type.

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

Inheritance diagram for ch_binary_semaphore:



Collaboration diagram for `ch_binary_semaphore`:



Additional Inherited Members

7.1.1 Detailed Description

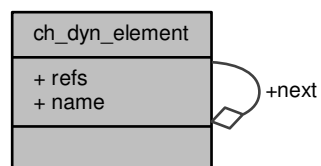
Binary semaphore type.

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

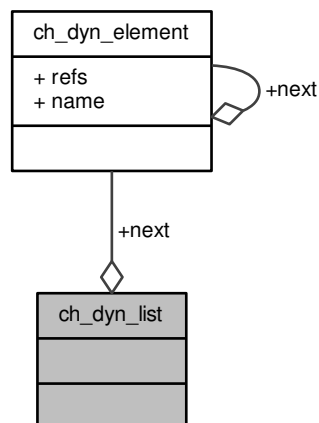
Number of references to this object.

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

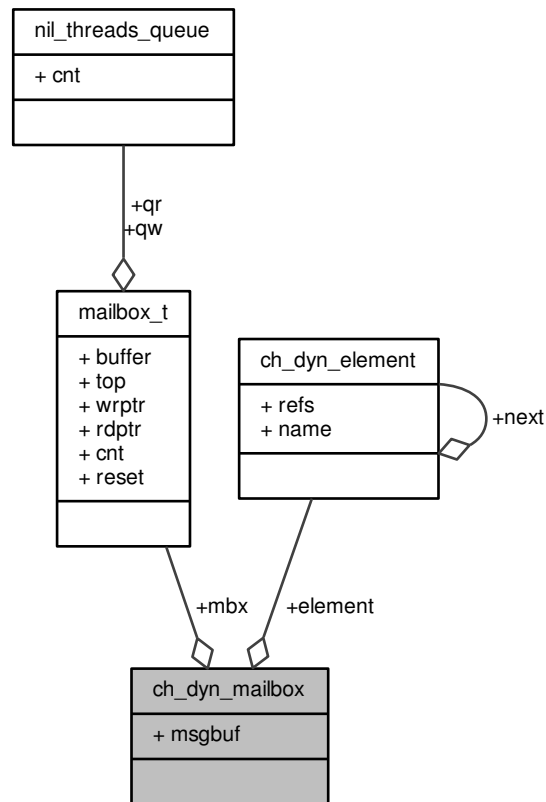
Type of a dynamic object list.

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

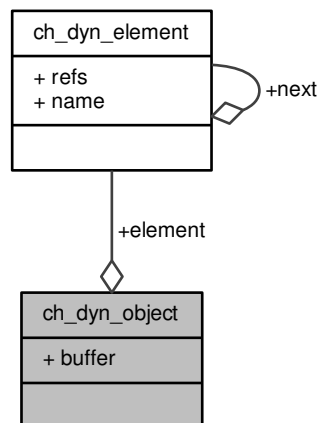
This requires C99.

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

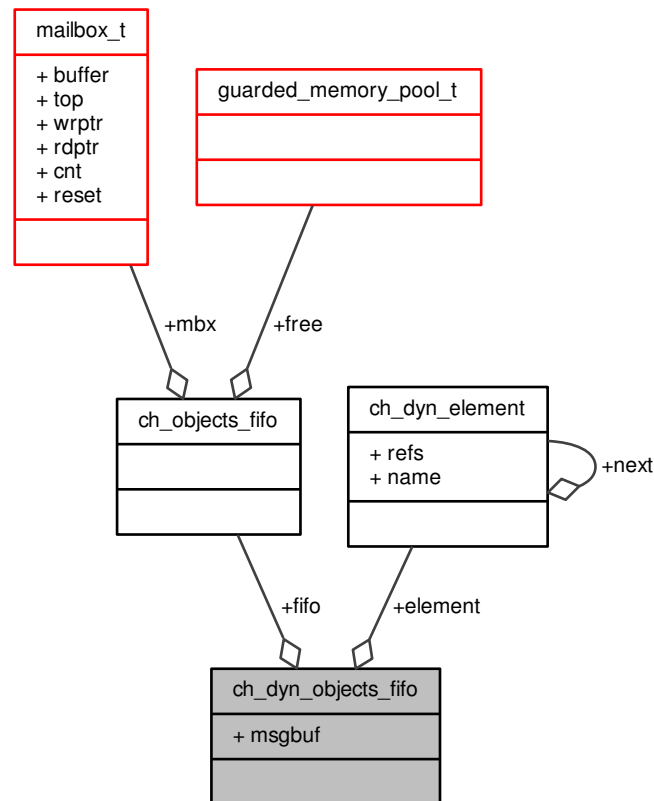
This requires C99.

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

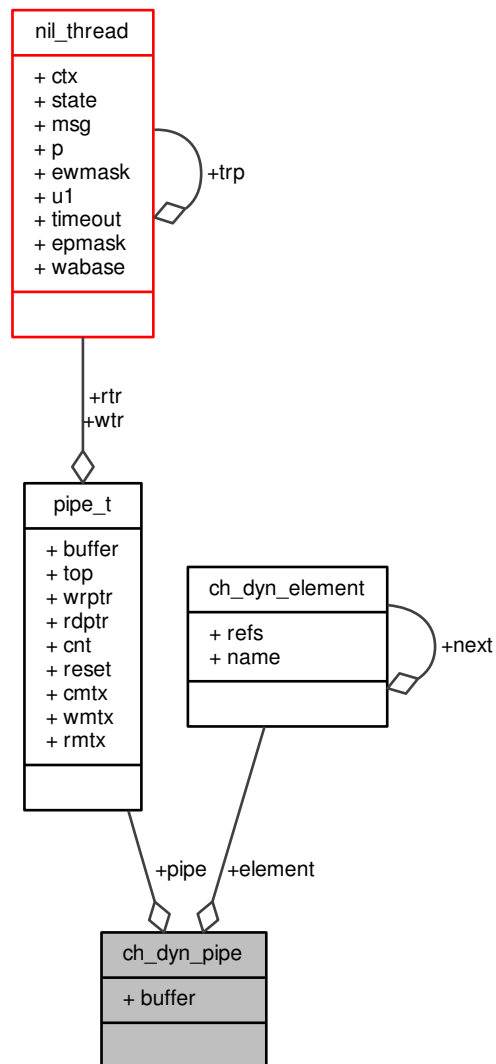
This open array is followed by another area containing the objects, this area is not represented in this structure. This requires C99.

7.7 `ch_dyn_pipe` Struct Reference

Type of a dynamic pipe object.

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


Collaboration diagram for ch_dyn_pipe:



Data Fields

- [dyn_element_t element](#)
List element of the dynamic pipe object.
- [pipe_t pipe](#)
The pipe.
- [uint8_t buffer \[\]](#)
Messages buffer.

7.7.1 Detailed Description

Type of a dynamic pipe object.

7.7.2 Field Documentation

7.7.2.1 `dyn_element_t ch_dyn_pipe::element`

List element of the dynamic pipe object.

7.7.2.2 `pipe_t ch_dyn_pipe::pipe`

The pipe.

7.7.2.3 `uint8_t ch_dyn_pipe::buffer[]`

Messages buffer.

Note

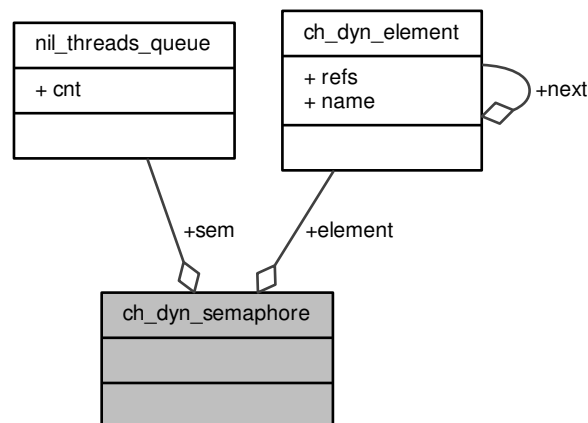
This requires C99.

7.8 `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.

7.8.1 Detailed Description

Type of a dynamic semaphore.

7.8.2 Field Documentation

7.8.2.1 dyn_element_t ch_dyn_semaphore::element

List element of the dynamic semaphore.

7.8.2.2 semaphore_t ch_dyn_semaphore::sem

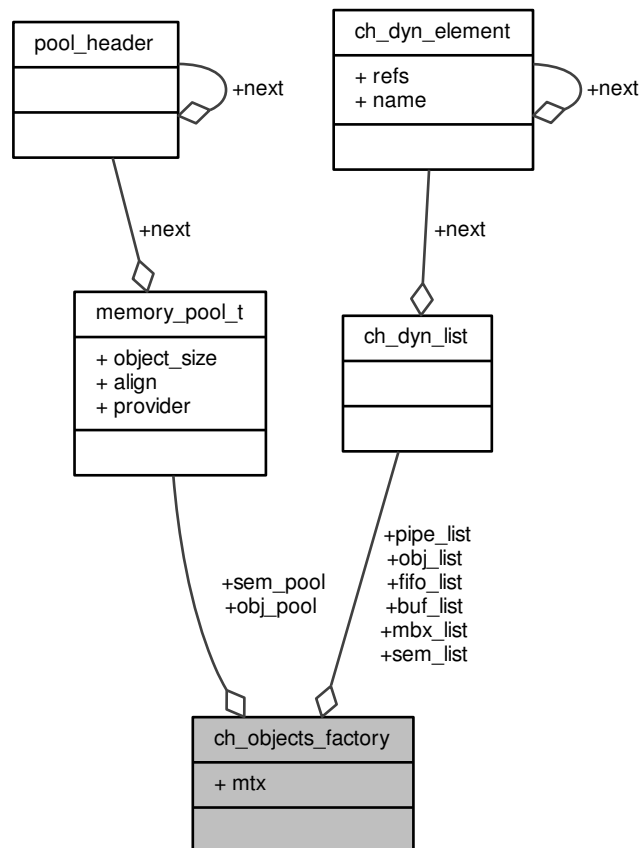
The semaphore.

7.9 ch_objects_factory Struct Reference

Type of the factory main object.

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

Collaboration diagram for ch_objects_factory:



Data Fields

- [mutex_t mtx](#)
Factory access mutex or semaphore.
- [dyn_list_t obj_list](#)
List of the registered objects.
- [memory_pool_t obj_pool](#)
Pool of the available registered objects.
- [dyn_list_t buf_list](#)
List of the allocated buffer objects.
- [dyn_list_t sem_list](#)
List of the allocated semaphores.
- [memory_pool_t sem_pool](#)
Pool of the available semaphores.
- [dyn_list_t mbx_list](#)
List of the allocated buffer objects.
- [dyn_list_t fifo_list](#)
List of the allocated "objects FIFO" objects.
- [dyn_list_t pipe_list](#)
List of the allocated pipe objects.

7.9.1 Detailed Description

Type of the factory main object.

7.9.2 Field Documentation

7.9.2.1 [mutex_t ch_objects_factory::mtx](#)

Factory access mutex or semaphore.

7.9.2.2 [dyn_list_t ch_objects_factory::obj_list](#)

List of the registered objects.

7.9.2.3 [memory_pool_t ch_objects_factory::obj_pool](#)

Pool of the available registered objects.

7.9.2.4 [dyn_list_t ch_objects_factory::buf_list](#)

List of the allocated buffer objects.

7.9.2.5 [dyn_list_t ch_objects_factory::sem_list](#)

List of the allocated semaphores.

7.9.2.6 [memory_pool_t ch_objects_factory::sem_pool](#)

Pool of the available semaphores.

7.9.2.7 dyn_list_t ch_objects_factory::mbx_list

List of the allocated buffer objects.

7.9.2.8 dyn_list_t ch_objects_factory::fifo_list

List of the allocated "objects FIFO" objects.

7.9.2.9 dyn_list_t ch_objects_factory::pipe_list

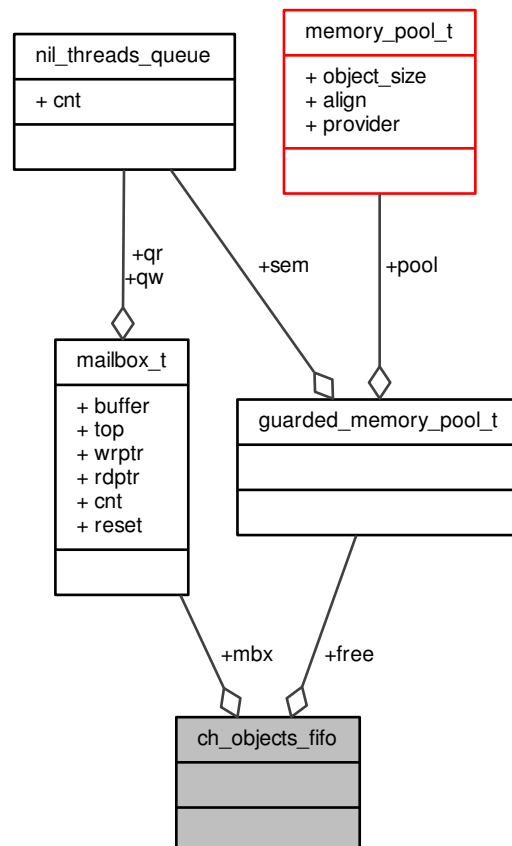
List of the allocated pipe objects.

7.10 ch_objects_fifo Struct Reference

Type of an objects FIFO.

```
#include <chobjfifos.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.

7.10.1 Detailed Description

Type of an objects FIFO.

7.10.2 Field Documentation

7.10.2.1 guarded_memory_pool_t ch_objects_fifo::free

Pool of the free objects.

7.10.2.2 mailbox_t ch_objects_fifo::mbx

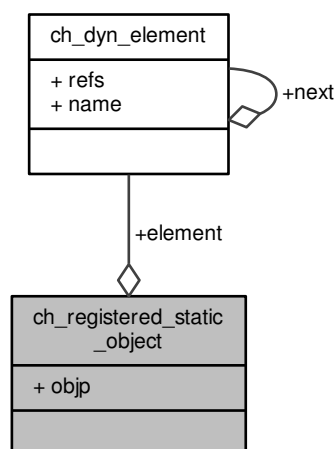
Mailbox of the sent objects.

7.11 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.

7.11.1 Detailed Description

Type of a registered object.

7.11.2 Field Documentation

7.11.2.1 [dyn_element_t ch_registered_static_object::element](#)

List element of the registered object.

7.11.2.2 [void* ch_registered_static_object::objp](#)

Pointer to the object.

Note

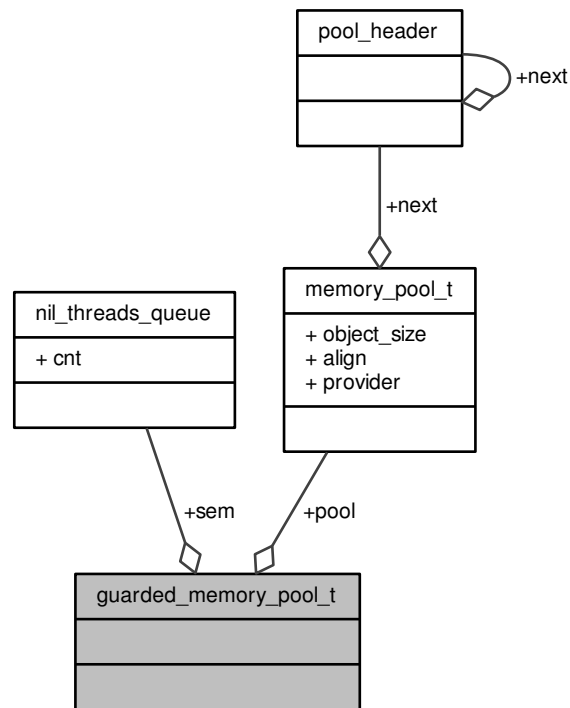
The type of the object is not stored in anyway.

7.12 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.

7.12.1 Detailed Description

Guarded memory pool descriptor.

7.12.2 Field Documentation

7.12.2.1 semaphore_t guarded_memory_pool_t::sem

Counter semaphore guarding the memory pool.

7.12.2.2 memory_pool_t guarded_memory_pool_t::pool

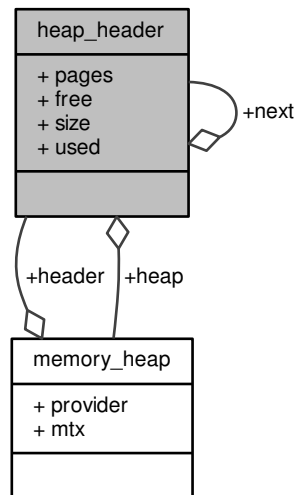
The memory pool itself.

7.13 heap_header Union Reference

Memory heap block header.

```
#include <chmemheaps.h>
```

Collaboration diagram for heap_header:



7.13.1 Detailed Description

Memory heap block header.

7.13.2 Field Documentation

7.13.2.1 `heap_header_t*` `heap_header::next`

Next block in free list.

7.13.2.2 `size_t` `heap_header::pages`

Size of the area in pages.

7.13.2.3 `memory_heap_t*` `heap_header::heap`

Block owner heap.

7.13.2.4 `size_t` `heap_header::size`

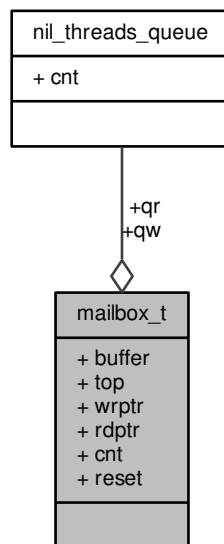
Size of the area in bytes.

7.14 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.

7.14.1 Detailed Description

Structure representing a mailbox object.

7.14.2 Field Documentation

7.14.2.1 msg_t* mailbox_t::buffer

Pointer to the mailbox buffer.

7.14.2.2 msg_t* mailbox_t::top

Pointer to the location after the buffer.

7.14.2.3 msg_t* mailbox_t::wrptr

Write pointer.

7.14.2.4 msg_t* mailbox_t::rdptr

Read pointer.

7.14.2.5 size_t mailbox_t::cnt

Messages in queue.

7.14.2.6 bool mailbox_t::reset

True in reset state.

7.14.2.7 threads_queue_t mailbox_t::qw

Queued writers.

7.14.2.8 threads_queue_t mailbox_t::qr

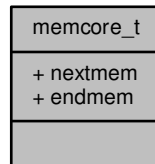
Queued readers.

7.15 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.

7.15.1 Detailed Description

Type of memory core object.

7.15.2 Field Documentation

7.15.2.1 `uint8_t* memcore_t::nextmem`

Next free address.

7.15.2.2 `uint8_t* memcore_t::endmem`

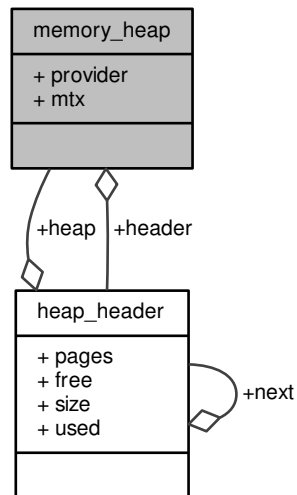
Final address.

7.16 memory_heap Struct Reference

Structure describing a memory heap.

```
#include <chmemheaps.h>
```

Collaboration diagram for memory_heap:



Data Fields

- [memgetfunc2_t provider](#)
Memory blocks provider for this heap.
- [heap_header_t header](#)
Free blocks list header.
- [mutex_t mtx](#)
Heap access mutex.

7.16.1 Detailed Description

Structure describing a memory heap.

7.16.2 Field Documentation

7.16.2.1 memgetfunc2_t memory_heap::provider

Memory blocks provider for this heap.

7.16.2.2 heap_header_t memory_heap::header

Free blocks list header.

7.16.2.3 mutex_t memory_heap::mtx

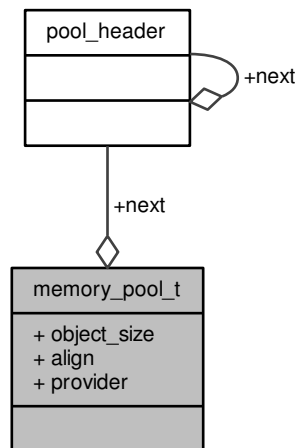
Heap access mutex.

7.17 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.

7.17.1 Detailed Description

Memory pool descriptor.

7.17.2 Field Documentation

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

Pointer to the header.

7.17.2.2 size_t [memory_pool_t::object_size](#)

Memory pool objects size.

7.17.2.3 unsigned memory_pool_t::align

Required alignment.

7.17.2.4 memgetfunc_t memory_pool_t::provider

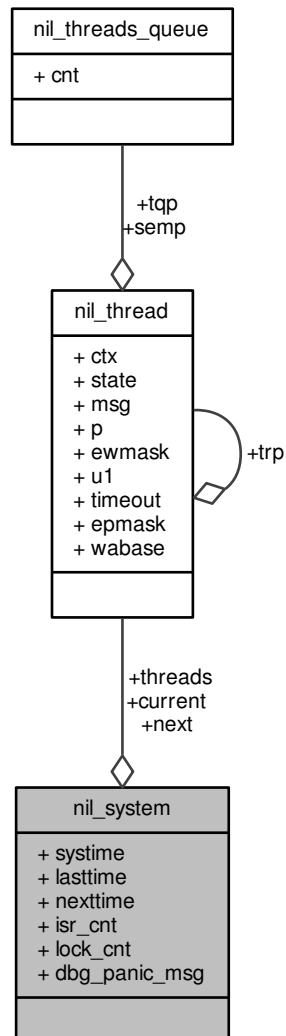
Memory blocks provider for this pool.

7.18 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 `system_t systime`
System time.
- `system_t lasttime`
System time of the last tick event.
- `system_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.

7.18.1 Detailed Description

System data structure.

Note

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

7.18.2 Field Documentation

7.18.2.1 `thread_t* nil_system::current`

Pointer to the running thread.

7.18.2.2 `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.

7.18.2.3 `volatile system_t nil_system::systime`

System time.

7.18.2.4 `system_t nil_system::lasttime`

System time of the last tick event.

7.18.2.5 `system_time_t nil_system::nexttime`

Time of the next scheduled tick event.

7.18.2.6 `cnt_t nil_system::isr_cnt`

ISR nesting level.

7.18.2.7 `cnt_t nil_system::lock_cnt`

Lock nesting level.

7.18.2.8 `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.

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

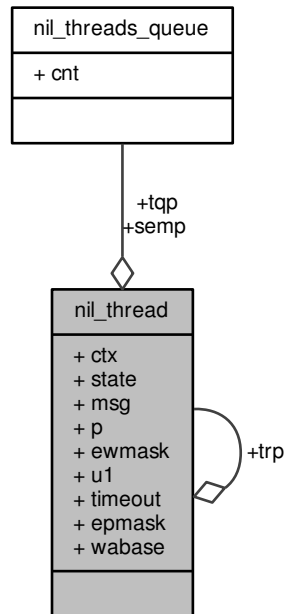
Thread structures for all the defined threads.

7.19 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.

7.19.1 Detailed Description

Structure representing a thread.

7.19.2 Field Documentation

7.19.2.1 struct port_context nil_thread::ctx

Processor context.

7.19.2.2 tstate_t nil_thread::state

Thread state.

7.19.2.3 msg_t nil_thread::msg

Wake-up message.

7.19.2.4 void* nil_thread::p

Generic pointer.

7.19.2.5 thread_reference_t* nil_thread::trp

Pointer to thread reference.

7.19.2.6 threads_queue_t* nil_thread::tqp

Pointer to thread queue.

7.19.2.7 semaphore_t* nil_thread::semp

Pointer to semaphore.

7.19.2.8 eventmask_t nil_thread::ewmask

Enabled events mask.

7.19.2.9 volatile sysinterval_t nil_thread::timeout

Timeout counter, zero if disabled.

7.19.2.10 eventmask_t nil_thread::epmask

Pending events mask.

7.19.2.11 stkalign_t* nil_thread::wabase

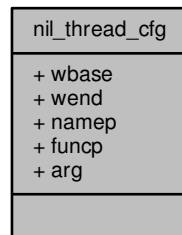
Thread stack boundary.

7.20 nil_thread_cfg Struct Reference

Structure representing a thread static configuration.

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

Collaboration diagram for nil_thread_cfg:



Data Fields

- `stkalignt * wbase`
Thread working area base.
- `stkalignt * wend`
Thread working area end.
- `const char * namep`
Thread name, for debugging.
- `tfunc_t funcp`
Thread function.
- `void * arg`
Thread function argument.

7.20.1 Detailed Description

Structure representing a thread static configuration.

7.20.2 Field Documentation

7.20.2.1 `stkalignt* nil_thread_cfg::wbase`

Thread working area base.

7.20.2.2 `stkalignt* nil_thread_cfg::wend`

Thread working area end.

7.20.2.3 `const char* nil_thread_cfg::namep`

Thread name, for debugging.

7.20.2.4 tfunc_t nil_thread_cfg::funcp

Thread function.

7.20.2.5 void* nil_thread_cfg::arg

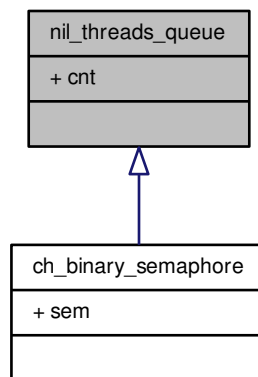
Thread function argument.

7.21 nil_threads_queue Struct Reference

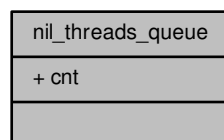
Structure representing a queue of threads.

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

Inheritance diagram for nil_threads_queue:



Collaboration diagram for nil_threads_queue:



Data Fields

- volatile cnt_t `cnt`

Threads Queue counter.

7.21.1 Detailed Description

Structure representing a queue of threads.

7.21.2 Field Documentation

7.21.2.1 volatile cnt_t nil_threads_queue::cnt

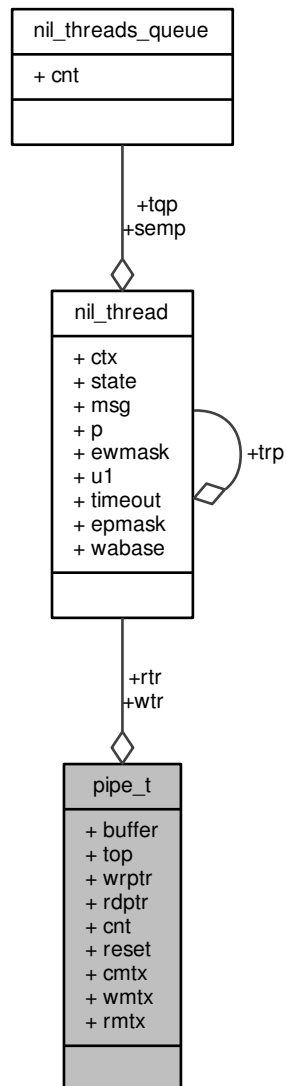
Threads Queue counter.

7.22 pipe_t Struct Reference

Structure representing a pipe object.

```
#include <chpipes.h>
```

Collaboration diagram for pipe_t:



Data Fields

- `uint8_t * buffer`
Pointer to the pipe buffer.
- `uint8_t * top`
Pointer to the location after the buffer.
- `uint8_t * wrptr`
Write pointer.
- `uint8_t * rdptr`
Read pointer.
- `size_t cnt`

Bytes in the pipe.

- bool `reset`

True if in reset state.

- `thread_reference_t wtr`

Waiting writer.

- `thread_reference_t rtr`

Waiting reader.

- `mutex_t cmtx`

Common access mutex.

- `mutex_t wmtx`

Write access mutex.

- `mutex_t rmtx`

Read access mutex.

7.22.1 Detailed Description

Structure representing a pipe object.

7.22.2 Field Documentation

7.22.2.1 `uint8_t* pipe_t::buffer`

Pointer to the pipe buffer.

7.22.2.2 `uint8_t* pipe_t::top`

Pointer to the location after the buffer.

7.22.2.3 `uint8_t* pipe_t::wrptr`

Write pointer.

7.22.2.4 `uint8_t* pipe_t::rdptr`

Read pointer.

7.22.2.5 `size_t pipe_t::cnt`

Bytes in the pipe.

7.22.2.6 `bool pipe_t::reset`

True if in reset state.

7.22.2.7 `thread_reference_t pipe_t::wtr`

Waiting writer.

7.22.2.8 thread_reference_t pipe_t::rtr

Waiting reader.

7.22.2.9 mutex_t pipe_t::cmtx

Common access mutex.

7.22.2.10 mutex_t pipe_t::wmtx

Write access mutex.

7.22.2.11 mutex_t pipe_t::rmtx

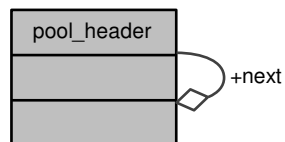
Read access mutex.

7.23 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.

7.23.1 Detailed Description

Memory pool free object header.

7.23.2 Field Documentation

7.23.2.1 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

- static [thread_t * nil_find_thread](#) (tstate_t state, void *p)
Retrieves the highest priority thread in the specified state and associated to the specified object.
- static cnt_t [nil_ready_all](#) (void *p, cnt_t cnt, msg_t msg)
Puts in ready state all thread matching the specified status and associated object.
- 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 [chThdResumel](#) ([thread_reference_t](#) *trp, [msg_t](#) msg)
Wakes up a thread waiting on a thread reference object.
- void [chThdResume](#) ([thread_reference_t](#) *trp, [msg_t](#) msg)
Wakes up a thread waiting on a thread reference object.
- void [chThdSleep](#) ([sysinterval_t](#) timeout)
Suspends the invoking thread for the specified time.
- void [chThdSleepUntil](#) ([system_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 [chSemResetl](#) ([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 [chEvtSignal1](#) ([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 "chlib.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` "3.2.2"
Kernel version string.
- #define `CH_KERNEL_MAJOR` 3
Kernel version major number.
- #define `CH_KERNEL_MINOR` 2
Kernel version minor number.
- #define `CH_KERNEL_PATCH` 2
Kernel version patch number.

Constants for configuration options

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

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`(tp) ((tp)->state == `NIL_STATE_READY`)
- #define `NIL_THD_IS_SLEEPING`(tp) ((tp)->state == `NIL_STATE_SLEEPING`)
- #define `NIL_THD_IS_SUSP`(tp) ((tp)->state == `NIL_STATE_SUSP`)
- #define `NIL_THD_IS_WTQUEUE`(tp) ((tp)->state == `NIL_STATE_WTQUEUE`)
- #define `NIL_THD_IS_WTOREVT`(tp) ((tp)->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`() `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 [chSchIsRescRequired\(\)](#) ((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.systemtime)
- *Current system time.*
#define [chVTTIMEElapsedSinceX](#)(start) [chTimeDiffX](#)((start), [chVTGetSystemTimeX](#)())
- *Returns the elapsed time since the specified start time.*
#define [chTimeAddX](#)(systemtime, interval) (([systemtime_t](#))(systemtime) + ([systemtime_t](#))(interval))
- *Adds an interval to a system time returning a system time.*
#define [chTimeDiffX](#)(start, end) (([sysinterval_t](#))(([systemtime_t](#))(end) - ([systemtime_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 [systemtime_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 [chSysTimerHandlerl](#) (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, [rtcnt_t](#) start, [rtcnt_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 [chThdResumel](#) ([thread_reference_t](#) *trp, [msg_t](#) msg)

- Wakes up a thread waiting on a thread reference object.*

 - void [chThdResume](#) ([thread_reference_t](#) *trp, [msg_t](#) msg)
- Wakes up a thread waiting on a thread reference object.*

 - 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 [chThdDoDequeueNextl](#) ([threads_queue_t](#) *tqp, [msg_t](#) msg)
- Dequeues and wakes up one thread from the threads queue object.*

 - void [chThdDequeueNextl](#) ([threads_queue_t](#) *tqp, [msg_t](#) msg)
- Dequeues and wakes up one thread from the threads queue object, if any.*

 - void [chThdDequeueAlll](#) ([threads_queue_t](#) *tqp, [msg_t](#) msg)
- Dequeues and wakes up all threads from the threads queue object.*

 - [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 [chSemSignal1](#) ([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 [chSemReset1](#) ([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 [chEvtSignal1](#) ([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 chBSemSignall (binary_semaphore_t *bsp)`
Performs a signal operation on a binary semaphore.
- `static void chBSemSignal (binary_semaphore_t *bsp)`
Performs a signal operation on a binary semaphore.
- `static bool chBSemGetStatel (const binary_semaphore_t *bsp)`
Returns the binary semaphore current state.

8.3.1 Detailed Description

Binary semaphores structures and macros.

Binary semaphores related APIs and services.

Operation mode

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

- **Taken**, when its counter has a value of zero or lower than zero. A negative number represent the number of threads queued on the binary semaphore.

- **Not taken**, when its counter has a value of one.

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

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

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_USE_PIPES TRUE`
Pipes 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.
- `#define CH_CFG_FACTORY_PIPES TRUE`
Enables factory for Pipes.

Debug options

- `#define CH_DBG_STATISTICS FALSE`
Debug option, kernel statistics.
- `#define CH_DBG_SYSTEM_STATE_CHECK TRUE`
Debug option, system state check.
- `#define CH_DBG_ENABLE_CHECKS TRUE`
Debug option, parameters checks.
- `#define CH_DBG_ENABLE_ASSERTS TRUE`
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.
- `dyn_pipe_t * chFactoryCreatePipe (const char *name, size_t size)`
Creates a dynamic pipe object.
- `dyn_pipe_t * chFactoryFindPipe (const char *name)`
Retrieves a dynamic pipe object.
- `void chFactoryReleasePipe (dyn_pipe_t *dpp)`
Releases a dynamic pipe object.

Variables

- `objects_factory_t ch_factory`
Factory object static instance.

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_dyn_pipe](#)
Type of a dynamic pipe object.
- struct [ch_objects_factory](#)
Type of the factory main object.

Macros

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

- `#define CH_CFG_FACTORY_MAILBOXES FALSE`
Enables factory for mailboxes.
- `#define CH_CFG_FACTORY_OBJ_FIFOS FALSE`
Enables factory for objects FIFOs.
- `#define CH_CFG_FACTORY_PIPES FALSE`
Enables factory for Pipes.

Typedefs

- typedef struct `ch_dyn_element` `dyn_element_t`
Type of a dynamic object list element.
- typedef struct `ch_dyn_list` `dyn_list_t`
Type of a dynamic object list.
- typedef struct `ch_registered_static_object` `registered_object_t`
Type of a registered object.
- typedef struct `ch_dyn_object` `dyn_buffer_t`
Type of a dynamic buffer object.
- typedef struct `ch_dyn_semaphore` `dyn_semaphore_t`
Type of a dynamic semaphore.
- typedef struct `ch_dyn_mailbox` `dyn_mailbox_t`
Type of a dynamic buffer object.
- typedef struct `ch_dyn_objects_fifo` `dyn_objects_fifo_t`
Type of a dynamic buffer object.
- typedef struct `ch_dyn_pipe` `dyn_pipe_t`
Type of a dynamic pipe object.
- typedef struct `ch_objects_factory` `objects_factory_t`
Type of the factory main object.

Functions

- void `_factory_init` (void)
Initializes the objects factory.
- `registered_object_t * chFactoryRegisterObject` (const char *name, void *objp)
Registers a generic object.
- `registered_object_t * chFactoryFindObject` (const char *name)
Retrieves a registered object.
- `registered_object_t * chFactoryFindObjectByPointer` (void *objp)
Retrieves a registered object by pointer.
- void `chFactoryReleaseObject` (`registered_object_t *rop`)
Releases a registered object.
- `dyn_buffer_t * chFactoryCreateBuffer` (const char *name, `size_t` size)
Creates a generic dynamic buffer object.
- `dyn_buffer_t * chFactoryFindBuffer` (const char *name)
Retrieves a dynamic buffer object.
- void `chFactoryReleaseBuffer` (`dyn_buffer_t *dbp`)
Releases a dynamic buffer object.
- `dyn_semaphore_t * chFactoryCreateSemaphore` (const char *name, `cnt_t` n)
Creates a dynamic semaphore object.
- `dyn_semaphore_t * chFactoryFindSemaphore` (const char *name)
Retrieves a dynamic semaphore object.

- void `chFactoryReleaseSemaphore` (`dyn_semaphore_t *dsp`)
Releases a dynamic semaphore object.
- `dyn_mailbox_t * chFactoryCreateMailbox` (`const char *name, size_t n`)
Creates a dynamic mailbox object.
- `dyn_mailbox_t * chFactoryFindMailbox` (`const char *name`)
Retrieves a dynamic mailbox object.
- void `chFactoryReleaseMailbox` (`dyn_mailbox_t *dmp`)
Releases a dynamic mailbox object.
- `dyn_objects_fifo_t * chFactoryCreateObjectsFIFO` (`const char *name, size_t objsize, size_t objn, unsigned objalign`)
Creates a dynamic "objects FIFO" object.
- `dyn_objects_fifo_t * chFactoryFindObjectsFIFO` (`const char *name`)
Retrieves a dynamic "objects FIFO" object.
- void `chFactoryReleaseObjectsFIFO` (`dyn_objects_fifo_t *dofp`)
Releases a dynamic "objects FIFO" object.
- `dyn_pipe_t * chFactoryCreatePipe` (`const char *name, size_t size`)
Creates a dynamic pipe object.
- `dyn_pipe_t * chFactoryFindPipe` (`const char *name`)
Retrieves a dynamic pipe object.
- void `chFactoryReleasePipe` (`dyn_pipe_t *dpp`)
Releases a dynamic pipe object.
- static `dyn_element_t * chFactoryDuplicateReference` (`dyn_element_t *dep`)
Duplicates an object reference.
- static void * `chFactoryGetObject` (`registered_object_t *rop`)
Returns the pointer to the inner registered object.
- static `size_t chFactoryGetBufferSize` (`dyn_buffer_t *dbp`)
Returns the size of a generic dynamic buffer object.
- static `uint8_t * chFactoryGetBuffer` (`dyn_buffer_t *dbp`)
Returns the pointer to the inner buffer.
- static `semaphore_t * chFactoryGetSemaphore` (`dyn_semaphore_t *dsp`)
Returns the pointer to the inner semaphore.
- static `mailbox_t * chFactoryGetMailbox` (`dyn_mailbox_t *dmp`)
Returns the pointer to the inner mailbox.
- static `objects_fifo_t * chFactoryGetObjectsFIFO` (`dyn_objects_fifo_t *dofp`)
Returns the pointer to the inner objects FIFO.
- static `pipe_t * chFactoryGetPipe` (`dyn_pipe_t *dpp`)
Returns the pointer to the inner pipe.

8.6.1 Detailed Description

ChibiOS objects factory structures and macros.

8.7 chlib.h File Reference

ChibiOS/LIB main include file.

```
#include "chbsem.h"
#include "chmboxes.h"
#include "chmemcore.h"
#include "chmemheaps.h"
#include "chmempools.h"
#include "chobjfifos.h"
#include "chpipes.h"
#include "chfactory.h"
```

Macros

- `#define _CHIBIOS_OSLIB_`
ChibiOS/LIB identification macro.
- `#define CH_OSLIB_STABLE 1`
Stable release flag.

ChibiOS/LIB version identification

- `#define CH_OSLIB_VERSION "1.1.2"`
OS Library version string.
- `#define CH_OSLIB_MAJOR 1`
OS Library version major number.
- `#define CH_OSLIB_MINOR 1`
OS Library version minor number.
- `#define CH_OSLIB_PATCH 2`
OS Library version patch number.

8.7.1 Detailed Description

ChibiOS/LIB main include file.

This header includes all the required library headers. This file is meant to be included by `ch.h` not directly by user.

8.8 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 chMBPostl (mailbox_t *mbp, msg_t msg)`
Posts a message into a mailbox.
 - `msg_t chMBPostAheadTimeout (mailbox_t *mbp, msg_t msg, sysinterval_t timeout)`
Posts an high priority message into a mailbox.
 - `msg_t chMBPostAheadTimeoutS (mailbox_t *mbp, msg_t msg, sysinterval_t timeout)`
Posts an high priority message into a mailbox.
 - `msg_t chMBPostAheadl (mailbox_t *mbp, msg_t msg)`
Posts an high priority message into a mailbox.
 - `msg_t chMBFetchTimeout (mailbox_t *mbp, msg_t *msgp, sysinterval_t timeout)`
Retrieves a message from a mailbox.
 - `msg_t chMBFetchTimeoutS (mailbox_t *mbp, msg_t *msgp, sysinterval_t timeout)`
Retrieves a message from a mailbox.
 - `msg_t chMBFetchl (mailbox_t *mbp, msg_t *msgp)`
Retrieves a message from a mailbox.

8.8.1 Detailed Description

Mailboxes code.

8.9 chmboxes.h File Reference

Mailboxes macros and structures.

Data Structures

- struct `mailbox_t`
Structure representing a mailbox object.

Macros

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

Functions

- void `chMBObjectInit (mailbox_t *mbp, msg_t *buf, size_t n)`
Initializes a `mailbox_t` object.
- void `chMBReset (mailbox_t *mbp)`
Resets a `mailbox_t` object.
- void `chMBResetl (mailbox_t *mbp)`
Resets a `mailbox_t` object.
- `msg_t chMBPostTimeout (mailbox_t *mbp, msg_t msg, sysinterval_t timeout)`
Posts a message into a mailbox.
- `msg_t chMBPostTimeoutS (mailbox_t *mbp, msg_t msg, sysinterval_t timeout)`
Posts a message into a mailbox.
- `msg_t chMBPostl (mailbox_t *mbp, msg_t msg)`

- Posts a message into a mailbox.*

 - `msg_t chMBPostAheadTimeout` (`mailbox_t *mbp`, `msg_t msg`, `sysinterval_t timeout`)

Posts an high priority message into a mailbox.

 - `msg_t chMBPostAheadTimeoutS` (`mailbox_t *mbp`, `msg_t msg`, `sysinterval_t timeout`)

Posts an high priority message into a mailbox.

 - `msg_t 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.

8.9.1 Detailed Description

Mailboxes macros and structures.

8.10 chmemcore.c File Reference

Core memory manager code.

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

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.*

Variables

- `memcore_t ch_memcore`
 - Memory core descriptor.*

8.10.1 Detailed Description

Core memory manager code.

8.11 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.

8.11.1 Detailed Description

Core memory manager macros and structures.

8.12 chmemheaps.c File Reference

Memory heaps code.

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

Functions

- void [_heap_init](#) (void)
Initializes the default heap.
- void [chHeapObjectInit](#) ([memory_heap_t](#) *heapp, void *buf, [size_t](#) size)
Initializes a memory heap from a static memory area.
- void * [chHeapAllocAligned](#) ([memory_heap_t](#) *heapp, [size_t](#) size, unsigned align)
Allocates a block of memory from the heap by using the first-fit algorithm.
- void [chHeapFree](#) (void *p)
Frees a previously allocated memory block.
- [size_t](#) [chHeapStatus](#) ([memory_heap_t](#) *heapp, [size_t](#) *totalp, [size_t](#) *largestp)
Reports the heap status.

Variables

- static [memory_heap_t](#) [default_heap](#)
Default heap descriptor.

8.12.1 Detailed Description

Memory heaps code.

8.13 chmemheaps.h File Reference

Memory heaps macros and structures.

Data Structures

- union [heap_header](#)
Memory heap block header.
- struct [memory_heap](#)
Structure describing a memory heap.

Macros

- #define [CH_HEAP_ALIGNMENT](#) 8U
Minimum alignment used for heap.
- #define [CH_HEAP_AREA](#)(name, size)
Allocation of an aligned static heap buffer.

Typedefs

- typedef struct [memory_heap](#) [memory_heap_t](#)
Type of a memory heap.
- typedef union [heap_header](#) [heap_header_t](#)
Type of a memory heap header.

Functions

- void [_heap_init](#) (void)
Initializes the default heap.
- void [chHeapObjectInit](#) ([memory_heap_t](#) *heapp, void *buf, size_t size)
Initializes a memory heap from a static memory area.
- void * [chHeapAllocAligned](#) ([memory_heap_t](#) *heapp, size_t size, unsigned align)
Allocates a block of memory from the heap by using the first-fit algorithm.
- void [chHeapFree](#) (void *p)
Frees a previously allocated memory block.
- size_t [chHeapStatus](#) ([memory_heap_t](#) *heapp, size_t *totalp, size_t *largestp)
Reports the heap status.
- static void * [chHeapAlloc](#) ([memory_heap_t](#) *heapp, size_t size)
Allocates a block of memory from the heap by using the first-fit algorithm.
- static size_t [chHeapGetSize](#) (const void *p)
Returns the size of an allocated block.

8.13.1 Detailed Description

Memory heaps 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 * [chPoolAllocI](#) ([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 [chPoolFreeI](#) ([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.

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.
- 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 cnt_t [chGuardedPoolGetCounterI](#) ([guarded_memory_pool_t](#) *gmp)

Gets the count of objects in a guarded memory pool.
- static void * [chGuardedPoolAllocI](#) ([guarded_memory_pool_t](#) *gmp)

Allocates an object from a guarded memory pool.
- static void [chGuardedPoolFreeI](#) ([guarded_memory_pool_t](#) *gmp, void *objp)

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

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

Adds an object to a guarded memory pool.
- static void [chGuardedPoolAddI](#) ([guarded_memory_pool_t](#) *gmp, void *objp)

Adds an object to a guarded memory pool.
- static void [chGuardedPoolAddS](#) ([guarded_memory_pool_t](#) *gmp, void *objp)

Adds an object to a guarded memory pool.

8.15.1 Detailed Description

Memory Pools macros and structures.

8.16 chobjfifos.h File Reference

Objects FIFO structures and macros.

Data Structures

- struct [ch_objects_fifo](#)
- Type of an objects FIFO.*

Typedefs

- typedef struct `ch_objects_fifo` `objects_fifo_t`
Type of an objects FIFO.

Functions

- static void `chFifoObjectInitAligned` (`objects_fifo_t *ofp`, `size_t objsize`, `size_t objn`, `unsigned objalign`, `void *objbuf`, `msg_t *msgbuf`)
Initializes a FIFO object.
- static void `chFifoObjectInit` (`objects_fifo_t *ofp`, `size_t objsize`, `size_t objn`, `void *objbuf`, `msg_t *msgbuf`)
Initializes a FIFO object.
- static void * `chFifoTakeObjectI` (`objects_fifo_t *ofp`)
Allocates a free object.
- static void * `chFifoTakeObjectTimeoutS` (`objects_fifo_t *ofp`, `sysinterval_t timeout`)
Allocates a free object.
- static void * `chFifoTakeObjectTimeout` (`objects_fifo_t *ofp`, `sysinterval_t timeout`)
Allocates a free object.
- static void `chFifoReturnObjectI` (`objects_fifo_t *ofp`, `void *objp`)
Releases a fetched object.
- static void `chFifoReturnObjectS` (`objects_fifo_t *ofp`, `void *objp`)
Releases a fetched object.
- static void `chFifoReturnObject` (`objects_fifo_t *ofp`, `void *objp`)
Releases a fetched object.
- static void `chFifoSendObjectI` (`objects_fifo_t *ofp`, `void *objp`)
Posts an object.
- static void `chFifoSendObjectS` (`objects_fifo_t *ofp`, `void *objp`)
Posts an object.
- static void `chFifoSendObject` (`objects_fifo_t *ofp`, `void *objp`)
Posts an object.
- static void `chFifoSendObjectAheadI` (`objects_fifo_t *ofp`, `void *objp`)
Posts an high priority object.
- static void `chFifoSendObjectAheadS` (`objects_fifo_t *ofp`, `void *objp`)
Posts an high priority object.
- static void `chFifoSendObjectAhead` (`objects_fifo_t *ofp`, `void *objp`)
Posts an high priority object.
- static `msg_t` `chFifoReceiveObjectI` (`objects_fifo_t *ofp`, `void **objpp`)
Fetches an object.
- static `msg_t` `chFifoReceiveObjectTimeoutS` (`objects_fifo_t *ofp`, `void **objpp`, `sysinterval_t timeout`)
Fetches an object.
- static `msg_t` `chFifoReceiveObjectTimeout` (`objects_fifo_t *ofp`, `void **objpp`, `sysinterval_t timeout`)
Fetches an object.

8.16.1 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.

8.17 chpipes.c File Reference

Pipes code.

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

Functions

- static size_t [pipe_write](#) ([pipe_t](#) *pp, const uint8_t *bp, size_t n)
Non-blocking pipe write.
- static size_t [pipe_read](#) ([pipe_t](#) *pp, uint8_t *bp, size_t n)
Non-blocking pipe read.
- void [chPipeObjectInit](#) ([pipe_t](#) *pp, uint8_t *buf, size_t n)
Initializes a [mailbox_t](#) object.
- void [chPipeReset](#) ([pipe_t](#) *pp)
Resets a [pipe_t](#) object.
- size_t [chPipeWriteTimeout](#) ([pipe_t](#) *pp, const uint8_t *bp, size_t n, [sysinterval_t](#) timeout)
Pipe write with timeout.
- size_t [chPipeReadTimeout](#) ([pipe_t](#) *pp, uint8_t *bp, size_t n, [sysinterval_t](#) timeout)
Pipe read with timeout.

8.17.1 Detailed Description

Pipes code.

Byte pipes.

Operation mode

A pipe is an asynchronous communication mechanism.

Operations defined for mailboxes:

- **Write:** Writes a buffer of data in the pipe in FIFO order.
- **Read:** A buffer of data is read from the read and removed.
- **Reset:** The pipe is emptied and all the stored data is lost.

Precondition

In order to use the pipes APIs the `CH_CFG_USE_PIPES` option must be enabled in [chconf.h](#).

Note

Compatible with RT and NIL.

8.18 chpipes.h File Reference

Pipes macros and structures.

Data Structures

- struct [pipe_t](#)
Structure representing a pipe object.

Macros

- #define [_PIPE_DATA](#)(name, buffer, size)
Data part of a static pipe initializer.
- #define [PIPE_DECL](#)(name, buffer, size) [pipe_t](#) name = [_PIPE_DATA](#)(name, buffer, size)
Static pipe initializer.

Functions

- void [chPipeObjectInit](#) ([pipe_t](#) *pp, uint8_t *buf, size_t n)
Initializes a [mailbox_t](#) object.
- void [chPipeReset](#) ([pipe_t](#) *pp)
Resets a [pipe_t](#) object.
- size_t [chPipeWriteTimeout](#) ([pipe_t](#) *pp, const uint8_t *bp, size_t n, [sysinterval_t](#) timeout)
Pipe write with timeout.
- size_t [chPipeReadTimeout](#) ([pipe_t](#) *pp, uint8_t *bp, size_t n, [sysinterval_t](#) timeout)
Pipe read with timeout.
- static size_t [chPipeGetSize](#) (const [pipe_t](#) *pp)
Returns the pipe buffer size as number of bytes.
- static size_t [chPipeGetUsedCount](#) (const [pipe_t](#) *pp)
Returns the number of used byte slots into a pipe.
- static size_t [chPipeGetFreeCount](#) (const [pipe_t](#) *pp)
Returns the number of free byte slots into a pipe.
- static void [chPipeResume](#) ([pipe_t](#) *pp)
Terminates the reset state.

8.18.1 Detailed Description

Pipes macros and structures.

Index

- `_BSEMAPHORE_DATA`
 - Binary Semaphores, 74
 - `_CHIBIOS_NIL_`
 - API, 26
 - `_CHIBIOS_OSLIB_`
 - Version Numbers and Identification, 72
 - `_GUARDEDMEMORYPOOL_DATA`
 - Memory Pools, 119
 - `_MAILBOX_DATA`
 - Mailboxes, 83
 - `_MEMORYPOOL_DATA`
 - Memory Pools, 118
 - `_PIPE_DATA`
 - Pipes, 97
 - `_SEMAPHORE_DATA`
 - API, 36
 - `_THREADS_QUEUE_DATA`
 - API, 36
 - `_THREADS_QUEUE_DECL`
 - API, 36
 - `__CH_STRINGIFY`
 - API, 29
 - `_core_init`
 - Core Memory Manager, 106
 - `_dbg_check_disable`
 - API, 49
 - `_dbg_check_enable`
 - API, 50
 - `_dbg_check_enter_isr`
 - API, 52
 - `_dbg_check_leave_isr`
 - API, 52
 - `_dbg_check_lock`
 - API, 50
 - `_dbg_check_lock_from_isr`
 - API, 51
 - `_dbg_check_suspend`
 - API, 49
 - `_dbg_check_unlock`
 - API, 50
 - `_dbg_check_unlock_from_isr`
 - API, 51
 - `_factory_init`
 - Dynamic Objects Factory, 153
 - `_heap_init`
 - Memory Heaps, 113
- `ALL_EVENTS`
 - API, 28
 - API, 19
 - `_CHIBIOS_NIL_`, 26
 - `_SEMAPHORE_DATA`, 36
 - `_THREADS_QUEUE_DATA`, 36
 - `_THREADS_QUEUE_DECL`, 36
 - `__CH_STRINGIFY`, 29
 - `_dbg_check_disable`, 49
 - `_dbg_check_enable`, 50
 - `_dbg_check_enter_isr`, 52
 - `_dbg_check_leave_isr`, 52
 - `_dbg_check_lock`, 50
 - `_dbg_check_lock_from_isr`, 51
 - `_dbg_check_suspend`, 49
 - `_dbg_check_unlock`, 50
 - `_dbg_check_unlock_from_isr`, 51
 - `ALL_EVENTS`, 28
 - `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`, 28
 - `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`, 32
 - `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`, 53
 - `chEvtSignal`, 68
 - `chEvtSignalI`, 69
 - `chEvtWaitAnyTimeout`, 69
 - `chSchDoReschedule`, 58
 - `chSchGoSleepTimeoutS`, 59
 - `chSchIsPreemptionRequired`, 58
 - `chSchIsRescRequiredI`, 39
 - `chSchReadyI`, 57
 - `chSchRescheduleS`, 58
 - `chSemFastSignalI`, 43
 - `chSemFastWaitI`, 43
 - `chSemGetCounterI`, 44

- chSemObjectInit, 42
- chSemReset, 67
- chSemResetI, 67
- chSemSignal, 66
- chSemSignalI, 66
- chSemWait, 42
- chSemWaitTimeout, 64
- chSemWaitTimeoutS, 65
- chSemWaitS, 43
- chSysDisable, 37
- chSysEnable, 38
- chSysGetRealtimeCounterX, 37
- chSysGetStatusAndLockX, 55
- chSysHalt, 54
- chSysInit, 53
- chSysIsCounterWithinX, 56
- chSysLock, 38
- chSysLockFromISR, 38
- chSysPolledDelayX, 57
- chSysRestoreStatusX, 56
- chSysSuspend, 37
- chSysTimerHandlerI, 54
- chSysUnconditionalLock, 55
- chSysUnconditionalUnlock, 55
- chSysUnlock, 38
- chSysUnlockFromISR, 39
- chThdDequeueAllI, 64
- chThdDequeueNextI, 63
- chThdDoDequeueNextI, 63
- chThdEnqueueTimeoutS, 62
- chThdGetSelfX, 40
- chThdQueueIsEmptyI, 42
- chThdQueueObjectInit, 41
- chThdResume, 61
- chThdResumeI, 60
- chThdSleep, 61
- chThdSleepMicroseconds, 40
- chThdSleepMilliseconds, 40
- chThdSleepSeconds, 40
- chThdSleepUntil, 62
- chThdSleepUntilS, 41
- chThdSleepS, 41
- chThdSuspendTimeoutS, 59
- chTimeAddX, 45
- chTimeDiffX, 45
- chTimeIsInRangeX, 45
- chVTGetSystemTimeX, 44
- chVTimeElapsedSinceX, 44
- EVENT_MASK, 28
- FALSE, 26
- MEM_ALIGN_MASK, 29
- MEM_ALIGN_NEXT, 29
- MEM_ALIGN_PREV, 29
- MEM_IS_ALIGNED, 30
- MEM_IS_VALID_ALIGNMENT, 30
- MSG_OK, 27
- MSG_RESET, 27
- MSG_TIMEOUT, 27
- NIL_STATE_READY, 27
- NIL_STATE_SLEEPING, 27
- NIL_STATE_SUSP, 27
- NIL_STATE_WTOREVT, 28
- NIL_STATE_WTQUEUE, 27
- nil, 70
- nil_find_thread, 48
- nil_ready_all, 48
- nil_system_t, 48
- SEMAPHORE_DECL, 36
- semaphore_t, 48
- sysinterval_t, 47
- sysptime_t, 47
- THD_FUNCTION, 31
- THD_IDLE_BASE, 28
- THD_TABLE_BEGIN, 29
- THD_TABLE_ENTRY, 29
- THD_TABLE_END, 29
- THD_WORKING_AREA_SIZE, 30
- THD_WORKING_AREA, 30
- TIME_I2MS, 35
- TIME_I2US, 35
- TIME_I2S, 34
- TIME_IMMEDIATE, 27
- TIME_INFINITE, 27
- TIME_MAX_INTERVAL, 27
- TIME_MAX_SYSTIME, 27
- TIME_MS2I, 33
- TIME_S2I, 33
- TIME_US2I, 34
- TRUE, 26
- 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, 188
- arg
 - nil_thread_cfg, 195
- BSEMAPHORE_DECL
 - Binary Semaphores, 75
- Binary Semaphores, 74
 - _BSEMAPHORE_DATA, 74
 - BSEMAPHORE_DECL, 75
 - binary_semaphore_t, 75
 - chBSemGetStatel, 80
 - chBSemObjectInit, 75
 - chBSemReset, 79
 - chBSemResetI, 78
 - chBSemSignal, 80
 - chBSemSignalI, 79
 - chBSemWait, 75
 - chBSemWaitTimeout, 77
 - chBSemWaitTimeoutS, 76
 - chBSemWaitS, 76
- binary_semaphore_t

- Binary Semaphores, [75](#)
- buf_list
 - ch_objects_factory, [178](#)
- buffer
 - ch_dyn_object, [172](#)
 - ch_dyn_pipe, [176](#)
 - mailbox_t, [185](#)
 - pipe_t, [198](#)
- CH_CFG_FACTORY_GENERIC_BUFFERS
 - API, [28](#)
 - Configuration, [16](#)
 - Dynamic Objects Factory, [151](#)
- CH_CFG_FACTORY_MAILBOXES
 - API, [28](#)
 - Configuration, [16](#)
 - Dynamic Objects Factory, [151](#)
- CH_CFG_FACTORY_MAX_NAMES_LENGTH
 - API, [28](#)
 - Configuration, [16](#)
 - Dynamic Objects Factory, [151](#)
- CH_CFG_FACTORY_OBJ_FIFOS
 - API, [28](#)
 - Configuration, [16](#)
 - Dynamic Objects Factory, [151](#)
- CH_CFG_FACTORY_OBJECTS_REGISTRY
 - API, [28](#)
 - Configuration, [16](#)
 - Dynamic Objects Factory, [151](#)
- CH_CFG_FACTORY_PIPES
 - Configuration, [16](#)
 - Dynamic Objects Factory, [151](#), [152](#)
- CH_CFG_FACTORY_SEMAPHORES
 - API, [28](#)
 - Configuration, [16](#)
 - Dynamic Objects Factory, [151](#)
- CH_CFG_IDLE_ENTER_HOOK
 - Configuration, [18](#)
- CH_CFG_IDLE_LEAVE_HOOK
 - Configuration, [18](#)
- CH_CFG_MEMCORE_SIZE
 - Configuration, [15](#)
 - Core Memory Manager, [106](#)
- CH_CFG_NUM_THREADS
 - Configuration, [13](#)
- CH_CFG_ST_FREQUENCY
 - Configuration, [14](#)
- 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, [28](#)
 - Configuration, [16](#)
- 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_PIPES
 - Configuration, [15](#)
- CH_CFG_USE_SEMAPHORES
 - Configuration, [14](#)
- CH_DBG_ENABLE_ASSERTS
 - Configuration, [17](#)
- CH_DBG_ENABLE_CHECKS
 - Configuration, [17](#)
- CH_DBG_ENABLE_STACK_CHECK
 - Configuration, [17](#)
- CH_DBG_STATISTICS
 - Configuration, [16](#)
- CH_DBG_SYSTEM_STATE_CHECK
 - Configuration, [17](#)
- CH_FAST_IRQ_HANDLER
 - API, [32](#)
- CH_HEAP_ALIGNMENT
 - Memory Heaps, [113](#)
- CH_HEAP_AREA
 - Memory Heaps, [113](#)
- 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, [32](#)
- 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_OSLIB_MAJOR

- Version Numbers and Identification, [72](#)
- CH_OSLIB_MINOR
 - Version Numbers and Identification, [72](#)
- CH_OSLIB_PATCH
 - Version Numbers and Identification, [72](#)
- CH_OSLIB_STABLE
 - Version Numbers and Identification, [72](#)
- CH_OSLIB_VERSION
 - Version Numbers and Identification, [72](#)
- ch.c, [201](#)
- ch.h, [203](#)
- ch_binary_semaphore, [167](#)
- ch_dyn_element, [168](#)
 - next, [169](#)
 - refs, [169](#)
- ch_dyn_list, [169](#)
- ch_dyn_mailbox, [170](#)
 - element, [171](#)
 - mbx, [171](#)
 - msgbuf, [171](#)
- ch_dyn_object, [171](#)
 - buffer, [172](#)
 - element, [172](#)
- ch_dyn_objects_fifo, [172](#)
 - element, [173](#)
 - fifo, [173](#)
 - msgbuf, [174](#)
- ch_dyn_pipe, [174](#)
 - buffer, [176](#)
 - element, [176](#)
 - pipe, [176](#)
- ch_dyn_semaphore, [176](#)
 - element, [177](#)
 - sem, [177](#)
- ch_factory
 - Dynamic Objects Factory, [165](#)
- ch_memcore
 - Core Memory Manager, [111](#)
- ch_objects_factory, [177](#)
 - buf_list, [178](#)
 - fifo_list, [179](#)
 - mbx_list, [178](#)
 - mtx, [178](#)
 - obj_list, [178](#)
 - obj_pool, [178](#)
 - pipe_list, [179](#)
 - sem_list, [178](#)
 - sem_pool, [178](#)
- ch_objects_fifo, [179](#)
 - free, [180](#)
 - mbx, [180](#)
- ch_registered_static_object, [180](#)
 - element, [181](#)
 - objp, [181](#)
- chBSemGetStatel
 - Binary Semaphores, [80](#)
- chBSemObjectInit
 - Binary Semaphores, [75](#)
- chBSemReset
 - Binary Semaphores, [79](#)
- chBSemResetl
 - Binary Semaphores, [78](#)
- chBSemSignal
 - Binary Semaphores, [80](#)
- chBSemSignall
 - Binary Semaphores, [79](#)
- chBSemWait
 - Binary Semaphores, [75](#)
- chBSemWaitTimeout
 - Binary Semaphores, [77](#)
- chBSemWaitTimeoutS
 - Binary Semaphores, [76](#)
- chBSemWaitS
 - Binary Semaphores, [76](#)
- chCoreAlloc
 - Core Memory Manager, [110](#)
- chCoreAllocAligned
 - Core Memory Manager, [109](#)
- chCoreAllocAlignedWithOffset
 - Core Memory Manager, [107](#)
- chCoreAllocAlignedWithOffsetl
 - Core Memory Manager, [106](#)
- chCoreAllocAlignedI
 - Core Memory Manager, [108](#)
- chCoreAllocI
 - Core Memory Manager, [109](#)
- chCoreGetStatusX
 - Core Memory Manager, [108](#)
- chDbgAssert
 - API, [46](#)
- chDbgCheck
 - API, [46](#)
- chDbgCheckClassI
 - API, [52](#)
- chDbgCheckClassS
 - API, [53](#)
- chEvtSignal
 - API, [68](#)
- chEvtSignall
 - API, [69](#)
- chEvtWaitAnyTimeout
 - API, [69](#)
- chFactoryCreateBuffer
 - Dynamic Objects Factory, [155](#)
- chFactoryCreateMailbox
 - Dynamic Objects Factory, [157](#)
- chFactoryCreateObjectsFIFO
 - Dynamic Objects Factory, [159](#)
- chFactoryCreatePipe
 - Dynamic Objects Factory, [160](#)
- chFactoryCreateSemaphore
 - Dynamic Objects Factory, [156](#)
- chFactoryDuplicateReference
 - Dynamic Objects Factory, [162](#)
- chFactoryFindBuffer
 - Dynamic Objects Factory, [155](#)

- chFactoryFindMailbox
 - Dynamic Objects Factory, [158](#)
- chFactoryFindObject
 - Dynamic Objects Factory, [153](#)
- chFactoryFindObjectByPointer
 - Dynamic Objects Factory, [154](#)
- chFactoryFindObjectFIFO
 - Dynamic Objects Factory, [160](#)
- chFactoryFindPipe
 - Dynamic Objects Factory, [161](#)
- chFactoryFindSemaphore
 - Dynamic Objects Factory, [157](#)
- chFactoryGetBuffer
 - Dynamic Objects Factory, [163](#)
- chFactoryGetBufferSize
 - Dynamic Objects Factory, [163](#)
- chFactoryGetMailbox
 - Dynamic Objects Factory, [164](#)
- chFactoryGetObject
 - Dynamic Objects Factory, [162](#)
- chFactoryGetObjectsFIFO
 - Dynamic Objects Factory, [164](#)
- chFactoryGetPipe
 - Dynamic Objects Factory, [165](#)
- chFactoryGetSemaphore
 - Dynamic Objects Factory, [164](#)
- chFactoryRegisterObject
 - Dynamic Objects Factory, [153](#)
- chFactoryReleaseBuffer
 - Dynamic Objects Factory, [156](#)
- chFactoryReleaseMailbox
 - Dynamic Objects Factory, [159](#)
- chFactoryReleaseObject
 - Dynamic Objects Factory, [154](#)
- chFactoryReleaseObjectsFIFO
 - Dynamic Objects Factory, [160](#)
- chFactoryReleasePipe
 - Dynamic Objects Factory, [162](#)
- chFactoryReleaseSemaphore
 - Dynamic Objects Factory, [157](#)
- chFifoObjectInit
 - Objects FIFOs, [136](#)
- chFifoObjectInitAligned
 - Objects FIFOs, [136](#)
- chFifoReceiveObjectTimeout
 - Objects FIFOs, [146](#)
- chFifoReceiveObjectTimeoutS
 - Objects FIFOs, [145](#)
- chFifoReceiveObjectI
 - Objects FIFOs, [144](#)
- chFifoReturnObject
 - Objects FIFOs, [140](#)
- chFifoReturnObjectI
 - Objects FIFOs, [139](#)
- chFifoReturnObjectS
 - Objects FIFOs, [140](#)
- chFifoSendObject
 - Objects FIFOs, [142](#)
- chFifoSendObjectAhead
 - Objects FIFOs, [144](#)
- chFifoSendObjectAheadI
 - Objects FIFOs, [142](#)
- chFifoSendObjectAheadS
 - Objects FIFOs, [143](#)
- chFifoSendObjectI
 - Objects FIFOs, [141](#)
- chFifoSendObjectS
 - Objects FIFOs, [141](#)
- chFifoTakeObjectTimeout
 - Objects FIFOs, [138](#)
- chFifoTakeObjectTimeoutS
 - Objects FIFOs, [138](#)
- chFifoTakeObjectI
 - Objects FIFOs, [137](#)
- chGuardedPoolAdd
 - Memory Pools, [131](#)
- chGuardedPoolAddI
 - Memory Pools, [132](#)
- chGuardedPoolAddS
 - Memory Pools, [133](#)
- chGuardedPoolAllocTimeout
 - Memory Pools, [125](#)
- chGuardedPoolAllocTimeoutS
 - Memory Pools, [124](#)
- chGuardedPoolAllocI
 - Memory Pools, [129](#)
- chGuardedPoolFree
 - Memory Pools, [126](#)
- chGuardedPoolFreeI
 - Memory Pools, [130](#)
- chGuardedPoolFreeS
 - Memory Pools, [131](#)
- chGuardedPoolGetCounterI
 - Memory Pools, [129](#)
- chGuardedPoolLoadArray
 - Memory Pools, [123](#)
- chGuardedPoolObjectInit
 - Memory Pools, [128](#)
- chGuardedPoolObjectInitAligned
 - Memory Pools, [123](#)
- chHeapAlloc
 - Memory Heaps, [115](#)
- chHeapAllocAligned
 - Memory Heaps, [114](#)
- chHeapFree
 - Memory Heaps, [115](#)
- chHeapGetSize
 - Memory Heaps, [116](#)
- chHeapObjectInit
 - Memory Heaps, [114](#)
- chHeapStatus
 - Memory Heaps, [115](#)
- chMBFetchTimeout
 - Mailboxes, [91](#)
- chMBFetchTimeoutS
 - Mailboxes, [92](#)

- chMBFetchI
 - Mailboxes, 93
- chMBGetFreeCountI
 - Mailboxes, 94
- chMBGetSizeI
 - Mailboxes, 94
- chMBGetUsedCountI
 - Mailboxes, 94
- chMBOBJECTInit
 - Mailboxes, 84
- chMBPeekI
 - Mailboxes, 95
- chMBPostAheadTimeout
 - Mailboxes, 88
- chMBPostAheadTimeoutS
 - Mailboxes, 89
- chMBPostAheadI
 - Mailboxes, 90
- chMBPostTimeout
 - Mailboxes, 85
- chMBPostTimeoutS
 - Mailboxes, 86
- chMBPostI
 - Mailboxes, 87
- chMBReset
 - Mailboxes, 84
- chMBResetI
 - Mailboxes, 85
- chMBResumeX
 - Mailboxes, 95
- chPipeGetFreeCount
 - Pipes, 103
- chPipeGetSize
 - Pipes, 102
- chPipeGetUsedCount
 - Pipes, 102
- chPipeObjectInit
 - Pipes, 99
- chPipeReadTimeout
 - Pipes, 101
- chPipeReset
 - Pipes, 100
- chPipeResume
 - Pipes, 103
- chPipeWriteTimeout
 - Pipes, 100
- chPoolAdd
 - Memory Pools, 127
- chPoolAddI
 - Memory Pools, 128
- chPoolAlloc
 - Memory Pools, 121
- chPoolAllocI
 - Memory Pools, 120
- chPoolFree
 - Memory Pools, 122
- chPoolFreeI
 - Memory Pools, 122
- chPoolLoadArray
 - Memory Pools, 120
- chPoolObjectInit
 - Memory Pools, 126
- chPoolObjectInitAligned
 - Memory Pools, 119
- chSchDoReschedule
 - API, 58
- chSchGoSleepTimeoutS
 - API, 59
- chSchIsPreemptionRequired
 - API, 58
- chSchIsRescRequiredI
 - API, 39
- chSchReadyI
 - API, 57
- chSchRescheduleS
 - API, 58
- chSemFastSignalI
 - API, 43
- chSemFastWaitI
 - API, 43
- chSemGetCounterI
 - API, 44
- chSemObjectInit
 - API, 42
- chSemReset
 - API, 67
- chSemResetI
 - API, 67
- chSemSignal
 - API, 66
- chSemSignalI
 - API, 66
- chSemWait
 - API, 42
- chSemWaitTimeout
 - API, 64
- chSemWaitTimeoutS
 - API, 65
- chSemWaitS
 - API, 43
- chSysDisable
 - API, 37
- chSysEnable
 - API, 38
- chSysGetRealtimeCounterX
 - API, 37
- chSysGetStatusAndLockX
 - API, 55
- chSysHalt
 - API, 54
- chSysInit
 - API, 53
- chSysIsCounterWithinX
 - API, 56
- chSysLock
 - API, 38

- chSysLockFromISR
 - API, [38](#)
- chSysPolledDelayX
 - API, [57](#)
- chSysRestoreStatusX
 - API, [56](#)
- chSysSuspend
 - API, [37](#)
- chSysTimerHandlerI
 - API, [54](#)
- chSysUnconditionalLock
 - API, [55](#)
- chSysUnconditionalUnlock
 - API, [55](#)
- chSysUnlock
 - API, [38](#)
- chSysUnlockFromISR
 - API, [39](#)
- chThdDequeueAllI
 - API, [64](#)
- chThdDequeueNextI
 - API, [63](#)
- chThdDoDequeueNextI
 - API, [63](#)
- chThdEnqueueTimeoutS
 - API, [62](#)
- chThdGetSelfX
 - API, [40](#)
- chThdQueueIsEmptyI
 - API, [42](#)
- chThdQueueObjectInit
 - API, [41](#)
- chThdResume
 - API, [61](#)
- chThdResumeI
 - API, [60](#)
- chThdSleep
 - API, [61](#)
- chThdSleepMicroseconds
 - API, [40](#)
- chThdSleepMilliseconds
 - API, [40](#)
- chThdSleepSeconds
 - API, [40](#)
- chThdSleepUntil
 - API, [62](#)
- chThdSleepUntilS
 - API, [41](#)
- chThdSleepS
 - API, [41](#)
- chThdSuspendTimeoutS
 - API, [59](#)
- chTimeAddX
 - API, [45](#)
- chTimeDiffX
 - API, [45](#)
- chTimeIsInRangeX
 - API, [45](#)
- chVTGetSystemTimeX
 - API, [44](#)
- chVTTimeElapsedSinceX
 - API, [44](#)
- chbsem.h, [209](#)
- chconf.h, [211](#)
- chfactory.c, [212](#)
- chfactory.h, [214](#)
- chlib.h, [216](#)
- chmboxes.c, [217](#)
- chmboxes.h, [218](#)
- chmemcore.c, [219](#)
- chmemcore.h, [220](#)
- chmemheaps.c, [221](#)
- chmemheaps.h, [221](#)
- chmempools.c, [222](#)
- chmempools.h, [223](#)
- chobjfifos.h, [224](#)
- chpipes.c, [226](#)
- chpipes.h, [227](#)
- cmtx
 - pipe_t, [199](#)
- cnt
 - mailbox_t, [185](#)
 - nil_threads_queue, [196](#)
 - pipe_t, [198](#)
- Complex Services, [134](#)
- 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_PIPES, [16](#)
 - CH_CFG_FACTORY_SEMAPHORES, [16](#)
 - CH_CFG_IDLE_ENTER_HOOK, [18](#)
 - CH_CFG_IDLE_LEAVE_HOOK, [18](#)
 - CH_CFG_MEMCORE_SIZE, [15](#)
 - CH_CFG_NUM_THREADS, [13](#)
 - CH_CFG_ST_FREQUENCY, [14](#)
 - 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, [16](#)
 - 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_PIPES, [15](#)
 - CH_CFG_USE_SEMAPHORES, [14](#)
 - CH_DBG_ENABLE_ASSERTS, [17](#)
 - CH_DBG_ENABLE_CHECKS, [17](#)

- CH_DBG_ENABLE_STACK_CHECK, 17
- CH_DBG_STATISTICS, 16
- CH_DBG_SYSTEM_STATE_CHECK, 17
- Core Memory Manager, 105
 - _core_init, 106
 - CH_CFG_MEMCORE_SIZE, 106
 - ch_memcore, 111
 - chCoreAlloc, 110
 - chCoreAllocAligned, 109
 - chCoreAllocAlignedWithOffset, 107
 - chCoreAllocAlignedWithOffsetI, 106
 - chCoreAllocAlignedI, 108
 - chCoreAllocI, 109
 - chCoreGetStatusX, 108
 - memgetfunc2_t, 106
 - memgetfunc_t, 106
- ctx
 - nil_thread, 193
- current
 - nil_system, 190
- dbg_panic_msg
 - nil_system, 191
- default_heap
 - Memory Heaps, 116
- dyn_buffer_t
 - Dynamic Objects Factory, 152
- dyn_element_t
 - Dynamic Objects Factory, 152
- dyn_list_t
 - Dynamic Objects Factory, 152
- dyn_mailbox_t
 - Dynamic Objects Factory, 152
- dyn_objects_fifo_t
 - Dynamic Objects Factory, 152
- dyn_pipe_t
 - Dynamic Objects Factory, 152
- dyn_semaphore_t
 - Dynamic Objects Factory, 152
- Dynamic Objects Factory, 148
 - _factory_init, 153
 - CH_CFG_FACTORY_GENERIC_BUFFERS, 151
 - CH_CFG_FACTORY_MAILBOXES, 151
 - CH_CFG_FACTORY_MAX_NAMES_LENGTH, 151
 - CH_CFG_FACTORY_OBJ_FIFOS, 151
 - CH_CFG_FACTORY_OBJECTS_REGISTRY, 151
 - CH_CFG_FACTORY_PIPES, 151, 152
 - CH_CFG_FACTORY_SEMAPHORES, 151
 - ch_factory, 165
 - chFactoryCreateBuffer, 155
 - chFactoryCreateMailbox, 157
 - chFactoryCreateObjectsFIFO, 159
 - chFactoryCreatePipe, 160
 - chFactoryCreateSemaphore, 156
 - chFactoryDuplicateReference, 162
 - chFactoryFindBuffer, 155
 - chFactoryFindMailbox, 158
 - chFactoryFindObject, 153
 - chFactoryFindObjectByPointer, 154
 - chFactoryFindObjectsFIFO, 160
 - chFactoryFindPipe, 161
 - chFactoryFindSemaphore, 157
 - chFactoryGetBuffer, 163
 - chFactoryGetBufferSize, 163
 - chFactoryGetMailbox, 164
 - chFactoryGetObject, 162
 - chFactoryGetObjectsFIFO, 164
 - chFactoryGetPipe, 165
 - chFactoryGetSemaphore, 164
 - chFactoryRegisterObject, 153
 - chFactoryReleaseBuffer, 156
 - chFactoryReleaseMailbox, 159
 - chFactoryReleaseObject, 154
 - chFactoryReleaseObjectsFIFO, 160
 - chFactoryReleasePipe, 162
 - chFactoryReleaseSemaphore, 157
 - dyn_buffer_t, 152
 - dyn_element_t, 152
 - dyn_list_t, 152
 - dyn_mailbox_t, 152
 - dyn_objects_fifo_t, 152
 - dyn_pipe_t, 152
 - dyn_semaphore_t, 152
 - objects_factory_t, 152
 - registered_object_t, 152
- EVENT_MASK
 - API, 28
- element
 - ch_dyn_mailbox, 171
 - ch_dyn_object, 172
 - ch_dyn_objects_fifo, 173
 - ch_dyn_pipe, 176
 - ch_dyn_semaphore, 177
 - ch_registered_static_object, 181
- endmem
 - memcore_t, 186
- epmask
 - nil_thread, 193
- ewmask
 - nil_thread, 193
- FALSE
 - API, 26
- fifo
 - ch_dyn_objects_fifo, 173
- fifo_list
 - ch_objects_factory, 179
- free
 - ch_objects_fifo, 180
- funcp
 - nil_thread_cfg, 194
- GUARDEDMEMORYPOOL_DECL
 - Memory Pools, 119
- guarded_memory_pool_t, 181
 - pool, 182

- sem, 182
- header
 - memory_heap, 187
- heap
 - heap_header, 183
- heap_header, 183
 - heap, 183
 - next, 183
 - pages, 183
 - size, 183
- heap_header_t
 - Memory Heaps, 113
- isr_cnt
 - nil_system, 191
- lasttime
 - nil_system, 190
- lock_cnt
 - nil_system, 191
- MAILBOX_DECL
 - Mailboxes, 83
- MEM_ALIGN_MASK
 - API, 29
- MEM_ALIGN_NEXT
 - API, 29
- MEM_ALIGN_PREV
 - API, 29
- MEM_IS_ALIGNED
 - API, 30
- MEM_IS_VALID_ALIGNMENT
 - API, 30
- MEMORYPOOL_DECL
 - Memory Pools, 119
- MSG_OK
 - API, 27
- MSG_RESET
 - API, 27
- MSG_TIMEOUT
 - API, 27
- mailbox_t, 184
 - buffer, 185
 - cnt, 185
 - qr, 185
 - qw, 185
 - rdptr, 185
 - reset, 185
 - top, 185
 - wrptr, 185
- Mailboxes, 82
 - _MAILBOX_DATA, 83
 - chMBFetchTimeout, 91
 - chMBFetchTimeoutS, 92
 - chMBFetchI, 93
 - chMBGetFreeCountI, 94
 - chMBGetSizeI, 94
 - chMBGetUsedCountI, 94
 - chMBOBJECTINIT, 84
 - chMBPeekI, 95
 - chMBPostAheadTimeout, 88
 - chMBPostAheadTimeoutS, 89
 - chMBPostAheadI, 90
 - chMBPostTimeout, 85
 - chMBPostTimeoutS, 86
 - chMBPostI, 87
 - chMBReset, 84
 - chMBResetI, 85
 - chMBResumeX, 95
 - MAILBOX_DECL, 83
- mbx
 - ch_dyn_mailbox, 171
 - ch_objects_fifo, 180
- mbx_list
 - ch_objects_factory, 178
- memcore_t, 185
 - endmem, 186
 - nextmem, 186
- memgetfunc2_t
 - Core Memory Manager, 106
- memgetfunc_t
 - Core Memory Manager, 106
- Memory Heaps, 112
 - _heap_init, 113
 - CH_HEAP_ALIGNMENT, 113
 - CH_HEAP_AREA, 113
 - chHeapAlloc, 115
 - chHeapAllocAligned, 114
 - chHeapFree, 115
 - chHeapGetSize, 116
 - chHeapObjectInit, 114
 - chHeapStatus, 115
 - default_heap, 116
 - heap_header_t, 113
 - memory_heap_t, 113
- Memory Management, 104
- Memory Pools, 117
 - _GUARDEDMEMORYPOOL_DATA, 119
 - _MEMORYPOOL_DATA, 118
 - chGuardedPoolAdd, 131
 - chGuardedPoolAddI, 132
 - chGuardedPoolAddS, 133
 - chGuardedPoolAllocTimeout, 125
 - chGuardedPoolAllocTimeoutS, 124
 - chGuardedPoolAllocI, 129
 - chGuardedPoolFree, 126
 - chGuardedPoolFreeI, 130
 - chGuardedPoolFreeS, 131
 - chGuardedPoolGetCounterI, 129
 - chGuardedPoolLoadArray, 123
 - chGuardedPoolObjectInit, 128
 - chGuardedPoolObjectInitAligned, 123
 - chPoolAdd, 127
 - chPoolAddI, 128
 - chPoolAlloc, 121
 - chPoolAllocI, 120

- chPoolFree, 122
- chPoolFreeI, 122
- chPoolLoadArray, 120
- chPoolObjectInit, 126
- chPoolObjectInitAligned, 119
- GUARDEDMEMORYPOOL_DECL, 119
- MEMORYPOOL_DECL, 119
- memory_heap, 186
 - header, 187
 - mtx, 187
 - provider, 187
- memory_heap_t
 - Memory Heaps, 113
- memory_pool_t, 188
 - align, 188
 - next, 188
 - object_size, 188
 - provider, 189
- msg
 - nil_thread, 193
- msgbuf
 - ch_dyn_mailbox, 171
 - ch_dyn_objects_fifo, 174
- mtx
 - ch_objects_factory, 178
 - memory_heap, 187
- NIL Kernel, 11
- NIL_STATE_READY
 - API, 27
- NIL_STATE_SLEEPING
 - API, 27
- NIL_STATE_SUSP
 - API, 27
- NIL_STATE_WTOREVT
 - API, 28
- NIL_STATE_WTQUEUE
 - API, 27
- namep
 - nil_thread_cfg, 194
- next
 - ch_dyn_element, 169
 - heap_header, 183
 - memory_pool_t, 188
 - nil_system, 190
 - pool_header, 199
- nextmem
 - memcore_t, 186
- nexttime
 - nil_system, 190
- nil
 - API, 70
- nil_find_thread
 - API, 48
- nil_ready_all
 - API, 48
- nil_system, 189
 - current, 190
 - dbg_panic_msg, 191
 - isr_cnt, 191
 - lasttime, 190
 - lock_cnt, 191
 - next, 190
 - nexttime, 190
 - system, 190
 - threads, 191
- nil_system_t
 - API, 48
- nil_thread, 191
 - ctx, 193
 - epmask, 193
 - ewmask, 193
 - msg, 193
 - p, 193
 - semp, 193
 - state, 193
 - timeout, 193
 - tqp, 193
 - trp, 193
 - wabase, 193
- nil_thread_cfg, 194
 - arg, 195
 - funcp, 194
 - namep, 194
 - wabase, 194
 - wend, 194
- nil_threads_queue, 195
 - cnt, 196
- OS Library, 71
- obj_list
 - ch_objects_factory, 178
- obj_pool
 - ch_objects_factory, 178
- object_size
 - memory_pool_t, 188
- Objects FIFOs, 135
 - chFifoObjectInit, 136
 - chFifoObjectInitAligned, 136
 - chFifoReceiveObjectTimeout, 146
 - chFifoReceiveObjectTimeoutS, 145
 - chFifoReceiveObjectI, 144
 - chFifoReturnObject, 140
 - chFifoReturnObjectI, 139
 - chFifoReturnObjectS, 140
 - chFifoSendObject, 142
 - chFifoSendObjectAhead, 144
 - chFifoSendObjectAheadI, 142
 - chFifoSendObjectAheadS, 143
 - chFifoSendObjectI, 141
 - chFifoSendObjectS, 141
 - chFifoTakeObjectTimeout, 138
 - chFifoTakeObjectTimeoutS, 138
 - chFifoTakeObjectI, 137
 - objects_fifo_t, 136
- objects_factory_t
 - Dynamic Objects Factory, 152
- objects_fifo_t

- Objects FIFOs, [136](#)
- objp
 - ch_registered_static_object, [181](#)
- p
 - nil_thread, [193](#)
- PIPE_DECL
 - Pipes, [98](#)
- pages
 - heap_header, [183](#)
- pipe
 - ch_dyn_pipe, [176](#)
- pipe_list
 - ch_objects_factory, [179](#)
- pipe_read
 - Pipes, [99](#)
- pipe_t, [196](#)
 - buffer, [198](#)
 - cmtx, [199](#)
 - cnt, [198](#)
 - rdptr, [198](#)
 - reset, [198](#)
 - rmtx, [199](#)
 - rtr, [198](#)
 - top, [198](#)
 - wmtx, [199](#)
 - wrptr, [198](#)
 - wtr, [198](#)
- pipe_write
 - Pipes, [98](#)
- Pipes, [97](#)
 - _PIPE_DATA, [97](#)
 - chPipeGetFreeCount, [103](#)
 - chPipeGetSize, [102](#)
 - chPipeGetUsedCount, [102](#)
 - chPipeObjectInit, [99](#)
 - chPipeReadTimeout, [101](#)
 - chPipeReset, [100](#)
 - chPipeResume, [103](#)
 - chPipeWriteTimeout, [100](#)
 - PIPE_DECL, [98](#)
 - pipe_read, [99](#)
 - pipe_write, [98](#)
- pool
 - guarded_memory_pool_t, [182](#)
- pool_header, [199](#)
 - next, [199](#)
- provider
 - memory_heap, [187](#)
 - memory_pool_t, [189](#)
- qr
 - mailbox_t, [185](#)
- qw
 - mailbox_t, [185](#)
- rdptr
 - mailbox_t, [185](#)
 - pipe_t, [198](#)
- refs
 - ch_dyn_element, [169](#)
- registered_object_t
 - Dynamic Objects Factory, [152](#)
- reset
 - mailbox_t, [185](#)
 - pipe_t, [198](#)
- rmtx
 - pipe_t, [199](#)
- rtr
 - pipe_t, [198](#)
- SEMAPHORE_DECL
 - API, [36](#)
- sem
 - ch_dyn_semaphore, [177](#)
 - guarded_memory_pool_t, [182](#)
- sem_list
 - ch_objects_factory, [178](#)
- sem_pool
 - ch_objects_factory, [178](#)
- semaphore_t
 - API, [48](#)
- semp
 - nil_thread, [193](#)
- size
 - heap_header, [183](#)
- state
 - nil_thread, [193](#)
- Synchronization, [73](#)
- sysinterval_t
 - API, [47](#)
- systeme
 - nil_system, [190](#)
- systeme_t
 - API, [47](#)
- THD_FUNCTION
 - API, [31](#)
- THD_IDLE_BASE
 - API, [28](#)
- THD_TABLE_BEGIN
 - API, [29](#)
- THD_TABLE_ENTRY
 - API, [29](#)
- 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, [27](#)

TIME_INFINITE
 API, [27](#)

TIME_MAX_INTERVAL
 API, [27](#)

TIME_MAX_SYSTIME
 API, [27](#)

TIME_MS2I
 API, [33](#)

TIME_S2I
 API, [33](#)

TIME_US2I
 API, [34](#)

TRUE
 API, [26](#)

tfunc_t
 API, [48](#)

thread_config_t
 API, [48](#)

thread_reference_t
 API, [48](#)

thread_t
 API, [47](#)

threads
 nil_system, [191](#)

threads_queue_t
 API, [47](#)

time_conv_t
 API, [47](#)

timeout
 nil_thread, [193](#)

top
 mailbox_t, [185](#)
 pipe_t, [198](#)

top
 nil_thread, [193](#)

trp
 nil_thread, [193](#)

Version Numbers and Identification, [72](#)
 _CHIBIOS_OSLIB_, [72](#)
 CH_OSLIB_MAJOR, [72](#)
 CH_OSLIB_MINOR, [72](#)
 CH_OSLIB_PATCH, [72](#)
 CH_OSLIB_STABLE, [72](#)
 CH_OSLIB_VERSION, [72](#)

wabase
 nil_thread, [193](#)

wbase
 nil_thread_cfg, [194](#)

wend
 nil_thread_cfg, [194](#)

wmtx
 pipe_t, [199](#)

wrptr
 mailbox_t, [185](#)
 pipe_t, [198](#)

wtr
 pipe_t, [198](#)