# ChibiOS/NIL

## 1.1 Copyright

## 1.2 Introduction

## 1.3 Related Documents

### 2 Module Index

2.1 Modules

### 3 Hierarchical Index

3.1 Class Hierarchy

### 4 Data Structure Index

4.1 Data Structures

### 5 File Index

5.1 File List

### 6 Module Documentation

6.1 NIL Kernel

#### 6.1.1 Detailed Description

6.2 Configuration

#### 6.2.1 Detailed Description

6.2.2 Macro Definition Documentation

##### 6.2.2.1 CH_CFG_MAX_THREADS

##### 6.2.2.2 CH_CFG_AUTOSTART_THREADS

##### 6.2.2.3 CH_CFG_ST_RESOLUTION

##### 6.2.2.4 CH_CFG_ST_FREQUENCY

##### 6.2.2.5 CH_CFG_ST_TIMEDELTA

##### 6.2.2.6 CH_CFG_USE_WAITEXIT

##### 6.2.2.7 CH_CFG_USE_SEMAPHORES

##### 6.2.2.8 CH_CFG_USE_MUTEXES

##### 6.2.2.9 CH_CFG_USE_EVENTS

##### 6.2.2.10 CH_CFG_USE_MESSAGES

##### 6.2.2.11 CH_CFG_USE_MAILBOXES

##### 6.2.2.12 CH_CFG_USE_MEMCORE

##### 6.2.2.13 CH_CFG_MEMCORE_SIZE

##### 6.2.2.14 CH_CFG_USE_HEAP

##### 6.2.2.15 CH_CFG_USE_MEMPOOLS

##### 6.2.2.16 CH_CFG_USE_OBJ_FIFOS

##### 6.2.2.17 CH_CFG_USE_PIPES

##### 6.2.2.18 CH_CFG_USE_OBJ_CACHES

##### 6.2.2.19 CH_CFG_USE_DELEGATES

##### 6.2.2.20 CH_CFG_USE_JOBS

##### 6.2.2.21 CH_CFG_USE_FACTORY
6.2.2.22 CH_CFG_FACTORY_MAX_NAMES_LENGTH ........................................ 20
6.2.2.23 CH_CFG_FACTORY_OBJECTS_REGISTRY ....................................... 20
6.2.2.24 CH_CFG_FACTORY_GENERIC_BUFFERS ........................................ 20
6.2.2.25 CH_CFG_FACTORY_SEMAPHORES ............................................... 20
6.2.2.26 CH_CFG_FACTORY_MAILBOXES ................................................. 20
6.2.2.27 CH_CFG_FACTORY_OBJ_FIFOS ................................................ 20
6.2.2.28 CH_CFG_FACTORY_PIPES ........................................................ 21
6.2.2.29 CH_DBG_STATISTICS .......................................................... 21
6.2.2.30 CH_DBG_SYSTEM_STATE_CHECK .............................................. 21
6.2.2.31 CH_DBG_ENABLE_CHECKS ....................................................... 21
6.2.2.32 CH_DBG_ENABLE_ASSERTS ....................................................... 22
6.2.2.33 CH_DBG_ENABLE_STACK_CHECK .............................................. 22
6.2.2.34 CH_CFG_SYSTEM_INIT_HOOK ................................................... 22
6.2.2.35 CH_CFG_THREAD_EXT_FIELDS ............................................... 22
6.2.2.36 CH_CFG_THREAD_EXT_INIT_HOOK ............................................ 22
6.2.2.37 CH_CFG_THREAD_EXIT_HOOK ................................................ 23
6.2.2.38 CH_CFG_IDLE_ENTER_HOOK .................................................. 23
6.2.2.39 CH_CFG_IDLE_LEAVE_HOOK ................................................... 23
6.2.2.40 CH_CFG_SYSTEM_HALT_HOOK ................................................ 23

6.3 Base API .................................................................................. 24
6.3.1 Detailed Description ............................................................. 24
6.3.2 Macro Definition Documentation ........................................... 31
   6.3.2.1 __CHIBIOS_NIL__ ................................................................. 31
   6.3.2.2 CH_KERNEL_STABLE .............................................................. 32
   6.3.2.3 CH_KERNEL_VERSION ........................................................... 32
   6.3.2.4 CH_KERNEL_MAJOR ............................................................... 32
   6.3.2.5 CH_KERNEL_MINOR .............................................................. 32
   6.3.2.6 CH_KERNEL_PATCH ............................................................... 32
   6.3.2.7 FALSE ............................................................................. 33
   6.3.2.8 TRUE ............................................................................... 33
   6.3.2.9 MSG_OK ........................................................................... 33
   6.3.2.10 MSG_TIMEOUT ................................................................. 33
   6.3.2.11 MSG_RESET ................................................................... 33
   6.3.2.12 TIME_IMMEDIATE ............................................................. 33
   6.3.2.13 TIME_INFINITE ............................................................... 34
   6.3.2.14 TIME_MAX_INTERVAL ...................................................... 34
   6.3.2.15 TIME_MAX_SYSTIME ....................................................... 34
   6.3.2.16 NIL_STATE_WTSTART ....................................................... 34
   6.3.2.17 NIL_STATE_READY ............................................................ 34
   6.3.2.18 NIL_STATE_SLEEPING ........................................................ 34
   6.3.2.19 NIL_STATE_SUSPENDED .................................................... 35
   6.3.2.20 NIL_STATE_WTEXIT ........................................................... 35
6.3.2.63 chSchGoSleepS ........................................ 50
6.3.2.64 chSchWakeupS ........................................ 51
6.3.2.65 chSchIsRescRequiredI ................................ 51
6.3.2.66 chThdGetSelfX ....................................... 52
6.3.2.67 chThdGetPriorityX .................................... 52
6.3.2.68 chThdResumeS ......................................... 52
6.3.2.69 chThdSleepSeconds .................................... 53
6.3.2.70 chThdSleepMilliseconds ................................ 53
6.3.2.71 chThdSleepMicroseconds ......................... 54
6.3.2.72 chThdSleepS .......................................... 54
6.3.2.73 chThdSleepUntilS .................................... 55
6.3.2.74 chThdQueueObjectInit .............................. 55
6.3.2.75 chThdQueuesEmptyI .................................. 55
6.3.2.76 chVTGetSystemTimeX ................................. 56
6.3.2.77 chVTTimeElapsedSinceX ............................. 56
6.3.2.78 chVTIsSystemTimeWithinX ............................ 57
6.3.2.79 chTimeAddX ........................................ 57
6.3.2.80 chTimeDiffX ........................................ 58
6.3.2.81 chDbgCheck .......................................... 59
6.3.2.82 chDbgAssert ......................................... 59

6.3.3 Typedef Documentation .................................. 59
  6.3.3.1 rtcnt_t ............................................... 60
  6.3.3.2 syssts_t ............................................ 60
  6.3.3.3 stkalign_t ......................................... 60
  6.3.3.4 tstate_t ............................................ 60
  6.3.3.5 tprio_t .............................................. 60
  6.3.3.6 msg_t ............................................... 60
  6.3.3.7 eventid_t .......................................... 60
  6.3.3.8 eventmask_t ........................................ 61
  6.3.3.9 eventflags_t ....................................... 61
  6.3.3.10 cnt_t .............................................. 61
  6.3.3.11 ucnt_t ............................................ 61
  6.3.3.12 systime_t ......................................... 61
  6.3.3.13 sysinterval_t ..................................... 61
  6.3.3.14 time_conv_t ...................................... 62
  6.3.3.15 os_instance_t ..................................... 62
  6.3.3.16 tfunc_t ............................................ 62
  6.3.3.17 thread_descriptor_t .............................. 62
  6.3.3.18 thread_t .......................................... 62
  6.3.3.19 thread_reference_t ............................... 63
  6.3.3.20 threads_queue_t .................................. 63
  6.3.3.21 semaphore_t ...................................... 63
6.3.4 Function Documentation .......................................................... 63
  6.3.4.1 nil_find_thread() .......................................................... 63
  6.3.4.2 nil_ready_all() ............................................................ 64
  6.3.4.3 __dbg_check_disable() .................................................... 65
  6.3.4.4 __dbg_check_suspend() ................................................... 65
  6.3.4.5 __dbg_check_enable() .................................................... 66
  6.3.4.6 __dbg_check_lock() ....................................................... 66
  6.3.4.7 __dbg_check_unlock() .................................................... 67
  6.3.4.8 __dbg_check_lock_from_isr() ........................................... 67
  6.3.4.9 __dbg_check_unlock_from_isr() ....................................... 68
  6.3.4.10 __dbg_check_enter_isr() .............................................. 68
  6.3.4.11 __dbg_check_leave_isr() ............................................. 69
  6.3.4.12 chDbgCheckClassI() ................................................... 69
  6.3.4.13 chDbgCheckClassS() ................................................... 70
  6.3.4.14 chSysInit() ............................................................. 70
  6.3.4.15 chSysHalt() ............................................................ 71
  6.3.4.16 chSysTimerHandlerI() ................................................ 72
  6.3.4.17 chSysUnconditionalLock() ......................................... 72
  6.3.4.18 chSysUnconditionalUnlock() ....................................... 73
  6.3.4.19 chSysGetStatusAndLockX() ......................................... 73
  6.3.4.20 chSysRestoreStatusX() .............................................. 73
  6.3.4.21 chSysIsCounterWithinX() .......................................... 74
  6.3.4.22 chSysPolledDelayX() ................................................ 75
  6.3.4.23 chSchReadyI() .......................................................... 75
  6.3.4.24 chSchIsPreemptionRequired() ...................................... 76
  6.3.4.25 chSchDoPreemption() ................................................ 77
  6.3.4.26 chSchRescheduleS() .................................................. 77
  6.3.4.27 chSchGoSleepTimeoutS() ............................................. 77
  6.3.4.28 chTimIsInRangeX() ................................................... 78
  6.3.4.29 chThdCreateI() ........................................................ 79
  6.3.4.30 chThdCreate() .......................................................... 80
  6.3.4.31 chThdExit() ............................................................. 81
  6.3.4.32 chThdWait() .............................................................. 82
  6.3.4.33 chThdSuspendTimeoutS() .............................................. 82
  6.3.4.34 chThdResumeI() ........................................................ 83
  6.3.4.35 chThdResume() .......................................................... 84
  6.3.4.36 chThdSleep() ............................................................ 84
  6.3.4.37 chThdSleepUntil() .................................................... 85
  6.3.4.38 chThdEnqueueTimeoutS() ............................................. 85
  6.3.4.39 chThdDequeueNextI() ................................................ 86
  6.3.4.40 chThdDequeueNextI() ................................................ 87
  6.3.4.41 chThdDequeueAllI() .................................................. 87
6.3.5 Variable Documentation .................................................. 88
6.3.5.1 nil ................................................................. 88
6.4 Semaphores ............................................................... 89
6.4.1 Detailed Description .................................................... 89
6.4.2 Macro Definition Documentation ..................................... 89
  6.4.2.1 __SEMAPHORE_DATA ........................................... 90
  6.4.2.2 SEMAPHORE_DECL ............................................. 90
  6.4.2.3 chSemObjectInit ............................................. 90
  6.4.2.4 chSemReset .................................................. 91
  6.4.2.5 chSemResetI ................................................ 91
  6.4.2.6 chSemWait .................................................. 92
  6.4.2.7 chSemWaitS ................................................ 92
  6.4.2.8 chSemFastWaitI ............................................. 93
  6.4.2.9 chSemFastSignalI .......................................... 93
  6.4.2.10 chSemGetCounterI ......................................... 94
6.4.3 Function Documentation ............................................... 94
  6.4.3.1 chSemWaitTimeout() ......................................... 94
  6.4.3.2 chSemWaitTimeoutS() ....................................... 95
  6.4.3.3 chSemSignal() .............................................. 96
  6.4.3.4 chSemSignalI() ............................................. 97
  6.4.3.5 chSemResetWithMessage() .................................. 97
  6.4.3.6 chSemResetWithMessageI() ................................ 98
6.5 Events ................................................................. 100
6.5.1 Detailed Description .................................................. 100
6.5.2 Macro Definition Documentation ..................................... 101
  6.5.2.1 ALL_EVENTS ................................................ 102
  6.5.2.2 EVENT_MASK ................................................ 102
  6.5.2.3 _EVENTSOURCE_DATA ...................................... 102
  6.5.2.4 EVENTSOURCE_DECL ...................................... 102
  6.5.2.5 chEvtObjectInit ........................................... 103
  6.5.2.6 chEvtRegisterMask ......................................... 103
  6.5.2.7 chEvtRegister ............................................... 104
  6.5.2.8 chEvtIsListeningI .......................................... 104
  6.5.2.9 chEvtBroadcast ............................................. 105
  6.5.2.10 chEvtBroadcastI .......................................... 105
  6.5.2.11 chEvtAddEventsI .......................................... 105
  6.5.2.12 chEvtGetEventsX .......................................... 106
  6.5.2.13 chEvtWaitOne ............................................. 106
  6.5.2.14 chEvtWaitAny ............................................. 107
  6.5.2.15 chEvtWaitAll ............................................. 108
6.5.3 Typedef Documentation ............................................... 108
  6.5.3.1 event_source_t ............................................. 108
6.5.3.2 evhandler_t .................................................. 108

6.5.4 Function Documentation .................................... 108
  6.5.4.1 chEvtRegisterMaskWithFlags() ...................... 109
  6.5.4.2 chEvtUnregister() ........................................ 109
  6.5.4.3 chEvtGetAndClearEvents() ............................. 110
  6.5.4.4 chEvtGetAndClearEventsI() ............................ 110
  6.5.4.5 chEvtAddEvents() ....................................... 111
  6.5.4.6 chEvtBroadcastFlags() .................................. 111
  6.5.4.7 chEvtGetAndClearFlags() ............................... 112
  6.5.4.8 chEvtSignal() ........................................... 113
  6.5.4.9 chEvtSignalI() .......................................... 114
  6.5.4.10 chEvtBroadcastFlagsI() ............................... 115
  6.5.4.11 chEvtGetAndClearFlagsI() ............................ 116
  6.5.4.12 chEvtDispatch() ........................................ 116
  6.5.4.13 chEvtWaitOneTimeout() ............................... 116
  6.5.4.14 chEvtWaitAnyTimeout() ............................... 117
  6.5.4.15 chEvtWaitAllTimeout() ............................... 118

6.6 Synchronous Messages ......................................... 120
  6.6.1 Detailed Description ..................................... 120
  6.6.2 Macro Definition Documentation ....................... 120
    6.6.2.1 chMsgWaitS ........................................... 120
    6.6.2.2 chMsgGet ............................................ 121
    6.6.2.3 chMsgReleaseS ....................................... 121
  6.6.3 Function Documentation .................................. 122
    6.6.3.1 chMsgSend() ......................................... 122
    6.6.3.2 chMsgWait() ........................................... 122
    6.6.3.3 chMsgWaitTimeout() .................................. 123
    6.6.3.4 chMsgWaitTimeoutS() ................................ 124
    6.6.3.5 chMsgRelease() ...................................... 125

6.7 OS Library .................................................... 126
  6.7.1 Detailed Description .................................... 126

6.8 Version Numbers and Identification ......................... 127
  6.8.1 Detailed Description .................................... 127
  6.8.2 Macro Definition Documentation ....................... 127
    6.8.2.1 __CHIBIOS_OSLIB__ .................................. 127
    6.8.2.2 CH_OSLIB_STABLE .................................... 127
    6.8.2.3 CH_OSLIB_VERSION ................................... 128
    6.8.2.4 CH_OSLIB_MAJOR ...................................... 128
    6.8.2.5 CH_OSLIB_MINOR ...................................... 128
    6.8.2.6 CH_OSLIB_PATCH ...................................... 128
  6.8.3 Function Documentation .................................. 128
    6.8.3.1 __oslib_init() ...................................... 129
6.9 Synchronization .......................................................... 130
  6.9.1 Detailed Description ............................................... 130
6.10 Binary Semaphores .................................................... 131
  6.10.1 Detailed Description ............................................. 131
  6.10.2 Macro Definition Documentation ................................ 131
    6.10.2.1 __BSEMAPHORE_DATA ....................................... 132
    6.10.2.2 BSEMAPHORE_DECL ......................................... 132
  6.10.3 Typedef Documentation ......................................... 132
    6.10.3.1 binary_semaphore_t ......................................... 132
  6.10.4 Function Documentation ........................................ 132
    6.10.4.1 chBSemObjectInit() ......................................... 133
    6.10.4.2 chBSemWait() .............................................. 133
    6.10.4.3 chBSemWaitS() ............................................. 134
    6.10.4.4 chBSemWaitTimeoutS() ..................................... 134
    6.10.4.5 chBSemWaitTimeout() .................................... 135
    6.10.4.6 chBSemReset() ............................................. 136
    6.10.4.7 chBSemResetI() ............................................ 137
    6.10.4.8 chBSemSignal() ............................................ 138
    6.10.4.9 chBSemSignalI() .......................................... 139
  6.11 Mailboxes ........................................................... 140
  6.11.1 Detailed Description ........................................... 140
  6.11.2 Macro Definition Documentation ................................ 141
    6.11.2.1 __MAILBOX_DATA ......................................... 142
    6.11.2.2 MAILBOX_DECL ........................................... 142
  6.11.3 Function Documentation ........................................ 142
    6.11.3.1 chMBObjectInit() .......................................... 143
    6.11.3.2 chMBReset() .............................................. 143
    6.11.3.3 chMBResetI() .............................................. 144
    6.11.3.4 chMBPostTimeout() ....................................... 145
    6.11.3.5 chMBPostTimeoutS() ...................................... 146
    6.11.3.6 chMBPost() ................................................. 147
    6.11.3.7 chMBPostAheadTimeout() .................................. 148
    6.11.3.8 chMBPostAheadTimeoutS() .................................. 149
    6.11.3.9 chMBPostAheadI() ......................................... 150
    6.11.3.10 chMBFetchTimeout() ..................................... 151
    6.11.3.11 chMBFetchTimeoutS() .................................... 152
    6.11.3.12 chMBFetchI() ............................................ 153
    6.11.3.13 chMBGetSizeI() .......................................... 154
    6.11.3.14 chMBGetUsedCountI() .................................... 154
    6.11.3.15 chMBGetFreeCountI() .................................... 155
    6.11.3.16 chMB PeekI() ............................................. 156
6.16.5 Variable Documentation .............................................. 201
  6.16.5.1 ch_memcore .................................................. 201
6.17 Memory Heaps .......................................................... 202
  6.17.1 Detailed Description ............................................ 202
  6.17.2 Macro Definition Documentation ................................ 203
    6.17.2.1 CH_HEAP_ALIGNMENT ...................................... 203
    6.17.2.2 CH_HEAP_AREA ......................................... 203
  6.17.3 Typedef Documentation ....................................... 204
    6.17.3.1 memory_heap_t .......................................... 204
    6.17.3.2 heap_header_t ......................................... 204
  6.17.4 Function Documentation ...................................... 204
    6.17.4.1 __heap_init() .......................................... 204
    6.17.4.2 chHeapObjectInit() ..................................... 204
    6.17.4.3 chHeapAllocAligned() ................................... 205
    6.17.4.4 chHeapFree() ........................................... 205
    6.17.4.5 chHeapStatus() ......................................... 206
    6.17.4.6 chHeapAlloc() .......................................... 206
    6.17.4.7 chHeapGetSize() ........................................ 207
  6.17.5 Variable Documentation ....................................... 208
    6.17.5.1 default_heap ........................................... 208
6.18 Memory Pools ................................................................ 209
  6.18.1 Detailed Description ............................................ 209
  6.18.2 Macro Definition Documentation ................................ 210
    6.18.2.1 __MEMORYPOOL_DATA .................................... 211
    6.18.2.2 MEMORYPOOL_DECL ...................................... 211
    6.18.2.3 __GUARDEDMEMORYPOOL_DATA .............................. 211
    6.18.2.4 GUARDEDMEMORYPOOL_DECL ............................... 212
  6.18.3 Function Documentation ...................................... 212
    6.18.3.1 chPoolObjectInitAligned() ................................ 212
    6.18.3.2 chPoolLoadArray() ....................................... 213
    6.18.3.3 chPoolAllocI() .......................................... 214
    6.18.3.4 chPoolAlloc() ........................................... 215
    6.18.3.5 chPoolFreeI() ........................................... 216
    6.18.3.6 chPoolFree() ........................................... 217
    6.18.3.7 chGuardedPoolObjectInitAligned() ....................... 217
    6.18.3.8 chGuardedPoolLoadArray() ................................ 218
    6.18.3.9 chGuardedPoolAllocTimeoutS() .......................... 219
    6.18.3.10 chGuardedPoolAllocTimeout() .......................... 220
    6.18.3.11 chGuardedPoolFree() .................................... 221
    6.18.3.12 chPoolObjectInit() ..................................... 222
    6.18.3.13 chPoolAdd() ........................................... 223
    6.18.3.14 chPoolAddI() .......................................... 223
6.22.4.8 chFactoryReleaseBuffer() ........................................... 268
6.22.4.9 chFactoryCreateSemaphore() ...................................... 270
6.22.4.10 chFactoryFindSemaphore() ....................................... 270
6.22.4.11 chFactoryReleaseSemaphore() ...................................... 271
6.22.4.12 chFactoryCreateMailbox() .......................................... 271
6.22.4.13 chFactoryFindMailbox() ............................................ 272
6.22.4.14 chFactoryReleaseMailbox() ......................................... 273
6.22.4.15 chFactoryCreateObjectsFIFO() ..................................... 273
6.22.4.16 chFactoryFindObjectObjectsFIFO() .............................. 274
6.22.4.17 chFactoryReleaseObjectsFIFO() .................................... 274
6.22.4.18 chFactoryCreatePipe() .............................................. 275
6.22.4.19 chFactoryFindPipe() ................................................ 276
6.22.4.20 chFactoryReleasePipe() ............................................. 276
6.22.4.21 chFactoryDuplicateReference() .................................... 277
6.22.4.22 chFactoryGetObject() ................................................ 277
6.22.4.23 chFactoryGetBufferSize() ......................................... 278
6.22.4.24 chFactoryGetBuffer() ............................................... 279
6.22.4.25 chFactoryGetSemaphore() .......................................... 280
6.22.4.26 chFactoryGetMailbox() .............................................. 280
6.22.4.27 chFactoryGetObjectsFIFO() ........................................ 280
6.22.4.28 chFactoryGetPipe() ................................................ 281

6.22.5 Variable Documentation ............................................. 281
6.22.5.1 ch_factory ......................................................... 281

7 Data Structure Documentation  ........................................ 283
7.1 ch_binary_semaphore Struct Reference ................................. 283
7.1.1 Detailed Description .................................................. 284
7.2 ch_dyn_element Struct Reference ...................................... 284
7.2.1 Detailed Description .................................................. 285
7.2.2 Field Documentation .................................................. 285
7.2.2.1 next ................................................................. 285
7.2.2.2 refs ................................................................. 285
7.3 ch_dyn_list Struct Reference .......................................... 285
7.3.1 Detailed Description .................................................. 286
7.4 ch_dyn_mailbox Struct Reference ..................................... 286
7.4.1 Detailed Description .................................................. 287
7.4.2 Field Documentation .................................................. 287
7.4.2.1 element ............................................................ 287
7.4.2.2 mbx ................................................................. 287
7.5 ch_dyn_object Struct Reference ...................................... 287
7.5.1 Detailed Description .................................................. 288
7.5.2 Field Documentation .................................................. 288
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.5.2.1</td>
<td>element</td>
<td>288</td>
</tr>
<tr>
<td>7.6</td>
<td>ch_dyn_objects_fifo Struct Reference</td>
<td>288</td>
</tr>
<tr>
<td>7.6.1</td>
<td>Detailed Description</td>
<td>289</td>
</tr>
<tr>
<td>7.6.2</td>
<td>Field Documentation</td>
<td>289</td>
</tr>
<tr>
<td>7.6.2.1</td>
<td>element</td>
<td>290</td>
</tr>
<tr>
<td>7.6.2.2</td>
<td>fifo</td>
<td>290</td>
</tr>
<tr>
<td>7.7</td>
<td>ch_dyn_pipe Struct Reference</td>
<td>290</td>
</tr>
<tr>
<td>7.7.1</td>
<td>Detailed Description</td>
<td>291</td>
</tr>
<tr>
<td>7.7.2</td>
<td>Field Documentation</td>
<td>292</td>
</tr>
<tr>
<td>7.7.2.1</td>
<td>element</td>
<td>292</td>
</tr>
<tr>
<td>7.7.2.2</td>
<td>pipe</td>
<td>292</td>
</tr>
<tr>
<td>7.8</td>
<td>ch_dyn_semaphore Struct Reference</td>
<td>292</td>
</tr>
<tr>
<td>7.8.1</td>
<td>Detailed Description</td>
<td>293</td>
</tr>
<tr>
<td>7.8.2</td>
<td>Field Documentation</td>
<td>293</td>
</tr>
<tr>
<td>7.8.2.1</td>
<td>element</td>
<td>293</td>
</tr>
<tr>
<td>7.8.2.2</td>
<td>sem</td>
<td>293</td>
</tr>
<tr>
<td>7.9</td>
<td>ch_job_descriptor Struct Reference</td>
<td>293</td>
</tr>
<tr>
<td>7.9.1</td>
<td>Detailed Description</td>
<td>294</td>
</tr>
<tr>
<td>7.9.2</td>
<td>Field Documentation</td>
<td>294</td>
</tr>
<tr>
<td>7.9.2.1</td>
<td>jobfunc</td>
<td>294</td>
</tr>
<tr>
<td>7.9.2.2</td>
<td>jobarg</td>
<td>294</td>
</tr>
<tr>
<td>7.10</td>
<td>ch_jobs_queue Struct Reference</td>
<td>294</td>
</tr>
<tr>
<td>7.10.1</td>
<td>Detailed Description</td>
<td>295</td>
</tr>
<tr>
<td>7.10.2</td>
<td>Field Documentation</td>
<td>295</td>
</tr>
<tr>
<td>7.10.2.1</td>
<td>free</td>
<td>296</td>
</tr>
<tr>
<td>7.10.2.2</td>
<td>mbx</td>
<td>296</td>
</tr>
<tr>
<td>7.11</td>
<td>ch_objects_cache Struct Reference</td>
<td>296</td>
</tr>
<tr>
<td>7.11.1</td>
<td>Detailed Description</td>
<td>298</td>
</tr>
<tr>
<td>7.11.2</td>
<td>Field Documentation</td>
<td>298</td>
</tr>
<tr>
<td>7.11.2.1</td>
<td>hashn</td>
<td>298</td>
</tr>
<tr>
<td>7.11.2.2</td>
<td>hashp</td>
<td>298</td>
</tr>
<tr>
<td>7.11.2.3</td>
<td>objn</td>
<td>298</td>
</tr>
<tr>
<td>7.11.2.4</td>
<td>objsz</td>
<td>299</td>
</tr>
<tr>
<td>7.11.2.5</td>
<td>objvp</td>
<td>299</td>
</tr>
<tr>
<td>7.11.2.6</td>
<td>lru</td>
<td>299</td>
</tr>
<tr>
<td>7.11.2.7</td>
<td>cache_sem</td>
<td>299</td>
</tr>
<tr>
<td>7.11.2.8</td>
<td>lru_sem</td>
<td>299</td>
</tr>
<tr>
<td>7.11.2.9</td>
<td>readf</td>
<td>299</td>
</tr>
<tr>
<td>7.11.2.10</td>
<td>writef</td>
<td>300</td>
</tr>
<tr>
<td>7.12</td>
<td>ch_objects_factory Struct Reference</td>
<td>300</td>
</tr>
<tr>
<td>7.12.1</td>
<td>Detailed Description</td>
<td>301</td>
</tr>
<tr>
<td>7.12.2</td>
<td>Field Documentation</td>
<td>301</td>
</tr>
</tbody>
</table>
7.24.1 Detailed Description ......................................................... 323
7.24.2 Field Documentation ....................................................... 323
  7.24.2.1 provider ................................................................. 323
  7.24.2.2 header ................................................................. 324
  7.24.2.3 mtx .............................................................. 324
7.25 memory_pool_t Struct Reference .................................................. 324
  7.25.1 Detailed Description ..................................................... 325
  7.25.2 Field Documentation ..................................................... 325
    7.25.2.1 next .............................................................. 325
    7.25.2.2 object_size ...................................................... 325
    7.25.2.3 align ............................................................ 325
    7.25.2.4 provider ......................................................... 326
7.26 nil_os_instance Struct Reference .................................................. 326
  7.26.1 Detailed Description ..................................................... 327
  7.26.2 Field Documentation ..................................................... 327
    7.26.2.1 current .......................................................... 327
    7.26.2.2 next .............................................................. 327
    7.26.2.3 systime ........................................................ 328
    7.26.2.4 lasttime ......................................................... 328
    7.26.2.5 nexttime ......................................................... 328
    7.26.2.6 isr_cnt ........................................................ 328
    7.26.2.7 lock_cnt ......................................................... 328
    7.26.2.8 dbg_panic_msg ............................................... 328
    7.26.2.9 threads ......................................................... 329
7.27 nil_thread Struct Reference ................................................... 329
  7.27.1 Detailed Description .................................................... 330
  7.27.2 Field Documentation .................................................... 330
    7.27.2.1 ctx ............................................................... 330
    7.27.2.2 state ............................................................ 331
    7.27.2.3 msg .............................................................. 331
    7.27.2.4 p ................................................................. 331
    7.27.2.5 nsp .............................................................. 331
    7.27.2.6 trp .............................................................. 331
    7.27.2.7 tqp .............................................................. 331
    7.27.2.8 tp ............................................................... 331
    7.27.2.9 semp ........................................................... 332
    7.27.2.10 ewmask ......................................................... 332
    7.27.2.11 timeout ......................................................... 332
    7.27.2.12 epmask ......................................................... 332
    7.27.2.13 stmsg .......................................................... 333
    7.27.2.14 wabase ........................................................ 333
7.28 nil_thread_descriptor Struct Reference ............................................. 333
<table>
<thead>
<tr>
<th>Section Reference</th>
<th>Detailed Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.28.1</td>
<td>334</td>
<td></td>
</tr>
<tr>
<td>7.28.2</td>
<td>334</td>
<td></td>
</tr>
<tr>
<td>7.28.2.1 name</td>
<td>334</td>
<td></td>
</tr>
<tr>
<td>7.28.2.2 wbase</td>
<td>334</td>
<td></td>
</tr>
<tr>
<td>7.28.2.3 wend</td>
<td>334</td>
<td></td>
</tr>
<tr>
<td>7.28.2.4 prio</td>
<td>335</td>
<td></td>
</tr>
<tr>
<td>7.28.2.5 funcp</td>
<td>335</td>
<td></td>
</tr>
<tr>
<td>7.28.2.6 arg</td>
<td>335</td>
<td></td>
</tr>
<tr>
<td>7.29</td>
<td>335</td>
<td></td>
</tr>
<tr>
<td>7.29.1</td>
<td>336</td>
<td></td>
</tr>
<tr>
<td>7.29.2</td>
<td>336</td>
<td></td>
</tr>
<tr>
<td>7.29.2.1 cnt</td>
<td>336</td>
<td></td>
</tr>
<tr>
<td>7.30</td>
<td>336</td>
<td></td>
</tr>
<tr>
<td>7.30.1</td>
<td>338</td>
<td></td>
</tr>
<tr>
<td>7.30.2</td>
<td>338</td>
<td></td>
</tr>
<tr>
<td>7.30.2.1 buffer</td>
<td>338</td>
<td></td>
</tr>
<tr>
<td>7.30.2.2 top</td>
<td>338</td>
<td></td>
</tr>
<tr>
<td>7.30.2.3 wrptr</td>
<td>339</td>
<td></td>
</tr>
<tr>
<td>7.30.2.4 rdptr</td>
<td>339</td>
<td></td>
</tr>
<tr>
<td>7.30.2.5 cnt</td>
<td>339</td>
<td></td>
</tr>
<tr>
<td>7.30.2.6 reset</td>
<td>339</td>
<td></td>
</tr>
<tr>
<td>7.30.2.7 wtr</td>
<td>339</td>
<td></td>
</tr>
<tr>
<td>7.30.2.8 rtr</td>
<td>339</td>
<td></td>
</tr>
<tr>
<td>7.30.2.9 cmtx</td>
<td>340</td>
<td></td>
</tr>
<tr>
<td>7.30.2.10 wmtx</td>
<td>340</td>
<td></td>
</tr>
<tr>
<td>7.30.2.11 rmtx</td>
<td>340</td>
<td></td>
</tr>
<tr>
<td>7.31</td>
<td>340</td>
<td></td>
</tr>
<tr>
<td>7.31.1</td>
<td>341</td>
<td></td>
</tr>
<tr>
<td>7.31.2</td>
<td>341</td>
<td></td>
</tr>
<tr>
<td>7.31.2.1 next</td>
<td>341</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Section Reference</th>
<th>Detailed Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>8.1 ch.c File Reference</td>
<td>343</td>
<td></td>
</tr>
<tr>
<td>8.1.1 Detailed Description</td>
<td>345</td>
<td></td>
</tr>
<tr>
<td>8.2 ch.h File Reference</td>
<td>345</td>
<td></td>
</tr>
<tr>
<td>8.2.1 Detailed Description</td>
<td>352</td>
<td></td>
</tr>
<tr>
<td>8.3 chbsem.h File Reference</td>
<td>352</td>
<td></td>
</tr>
<tr>
<td>8.3.1 Detailed Description</td>
<td>353</td>
<td></td>
</tr>
<tr>
<td>8.4 chconf.h File Reference</td>
<td>353</td>
<td></td>
</tr>
<tr>
<td>8.4.1 Detailed Description</td>
<td>355</td>
<td></td>
</tr>
<tr>
<td>8.5 chdelegates.c File Reference</td>
<td>355</td>
<td></td>
</tr>
<tr>
<td>8.5.1 Detailed Description</td>
<td>356</td>
<td></td>
</tr>
</tbody>
</table>
8.27 chpipes.h File Reference .............................................. 380
8.27.1 Detailed Description .............................................. 381
8.28 chsem.c File Reference .............................................. 381
8.28.1 Detailed Description .............................................. 382
8.29 chsem.h File Reference .............................................. 382
8.29.1 Detailed Description .............................................. 382

Index ................................................................. 383
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
Base API ............................................... 24
Semaphores ........................................... 89
Events .................................................. 100
Synchronous Messages ................................ 120
OS Library ............................................. 126
Version Numbers and Identification ............ 127
Synchronization ..................................... 130
Binary Semaphores .................................. 131
Mailboxes ............................................ 140
Pipes .................................................. 158
Delegate Threads .................................... 166
Jobs Queues .......................................... 178
Memory Management ................................ 190
Core Memory Manager .............................. 191
Memory Heaps ....................................... 202
Memory Pools ........................................ 209
Complex Services .................................... 232
Objects FIFOs ....................................... 233
Objects Caches ..................................... 249
Dynamic Objects Factory .......................... 258
Chapter 3

Hierarchical Index

3.1 Class Hierarchy

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

ch_dyn_element ......................................................... 284
ch_dyn_list ............................................................... 285
ch_dyn_mailbox ......................................................... 286
ch_dyn_object ........................................................... 287
ch_dyn_objects_fifo ..................................................... 288
ch_dyn_pipe ............................................................... 290
ch_dyn_semaphore ....................................................... 292
ch_job_descriptor ....................................................... 293
ch_jobs_queue ........................................................... 294
ch_objects_cache ......................................................... 296
ch_objects_factory ...................................................... 300
ch_objects_fifo .......................................................... 303
ch_oc_hash_header ...................................................... 304
ch_oc_lru_header ......................................................... 306
ch_oc_object ............................................................. 308
ch_registered_static_object .......................................... 311
event_listener ............................................................ 312
event_source ............................................................. 314
guarded_memory_pool_t .................................................. 316
heap_header ............................................................... 317
mailbox_t ................................................................. 319
memcore_t ................................................................. 321
memory_heap ............................................................. 322
memory_pool_t ........................................................... 324
nil_os_instance .......................................................... 326
nil_thread ................................................................. 329
nil_thread_descriptor ................................................... 333
nil_threads_queue ........................................................ 335
ch_binary_semaphore ..................................................... 283
pipe_t ..................................................................... 336
pool_header ............................................................... 340
Hierarchical Index

6

ChibiOS/NIL
Chapter 4

Data Structure Index

4.1 Data Structures

Here are the data structures with brief descriptions:

- **ch_binary_semaphore**
  Binary semaphore type .......................................................... 283
- **ch_dyn_element**
  Type of a dynamic object list element .................................... 284
- **ch_dyn_list**
  Type of a dynamic object list .................................................. 285
- **ch_dyn_mailbox**
  Type of a dynamic buffer object ............................................. 286
- **ch_dyn_object**
  Type of a dynamic buffer object ............................................. 287
- **ch_dyn_objects_fifo**
  Type of a dynamic buffer object ............................................. 288
- **ch_dyn_pipe**
  Type of a dynamic pipe object ............................................... 290
- **ch_dyn_semaphore**
  Type of a dynamic semaphore .................................................. 292
- **ch_job_descriptor**
  Type of a job descriptor .......................................................... 293
- **ch_jobs_queue**
  Type of a jobs queue .............................................................. 294
- **ch_objects_cache**
  Structure representing a cache object ...................................... 296
- **ch_objects_factory**
  Type of the factory main object .............................................. 300
- **ch_objects_fifo**
  Type of an objects FIFO .......................................................... 303
- **ch_oc_hash_header**
  Structure representing an hash table element ............................... 304
- **ch_oc_lru_header**
  Structure representing an hash table element ............................... 306
- **ch_oc_object**
  Structure representing a cached object ..................................... 308
- **ch_registered_static_object**
  Type of a registered object .................................................... 311
- **event_listener**
  Event Listener structure .......................................................... 312
<table>
<thead>
<tr>
<th>Data Structure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>event_source</td>
<td>Event Source structure</td>
<td>314</td>
</tr>
<tr>
<td>guarded_memory_pool_t</td>
<td>Guarded memory pool descriptor</td>
<td>316</td>
</tr>
<tr>
<td>heap_header</td>
<td>Memory heap block header</td>
<td>317</td>
</tr>
<tr>
<td>mailbox_t</td>
<td>Structure representing a mailbox object</td>
<td>319</td>
</tr>
<tr>
<td>memcore_t</td>
<td>Type of memory core object</td>
<td>321</td>
</tr>
<tr>
<td>memory_heap</td>
<td>Structure describing a memory heap</td>
<td>322</td>
</tr>
<tr>
<td>memory_pool_t</td>
<td>Memory pool descriptor</td>
<td>324</td>
</tr>
<tr>
<td>nil_os_instance</td>
<td>System data structure</td>
<td>326</td>
</tr>
<tr>
<td>nil_thread</td>
<td>Structure representing a thread</td>
<td>329</td>
</tr>
<tr>
<td>nil_thread_descriptor</td>
<td>Structure representing a thread descriptor</td>
<td>333</td>
</tr>
<tr>
<td>nil_threads_queue</td>
<td>Structure representing a queue of threads</td>
<td>335</td>
</tr>
<tr>
<td>pipe_t</td>
<td>Structure representing a pipe object</td>
<td>336</td>
</tr>
<tr>
<td>pool_header</td>
<td>Memory pool free object header</td>
<td>340</td>
</tr>
</tbody>
</table>
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 .................................................. 343
- **ch.h**: Nil RTOS main header file ................................................. 345
- **chbsem.h**: Binary semaphores structures and macros .................. 352
- **chconf.h**: Configuration file template ....................................... 353
- **chdelegates.c**: Delegate threads code ..................................... 355
- **chdelegates.h**: Delegate threads macros and structures ............... 356
- **chevt.c**: Nil RTOS events source file ....................................... 357
- **chevt.h**: Nil RTOS events header file ...................................... 358
- **chfactory.c**: ChibiOS objects factory and registry code ............... 360
- **chfactory.h**: ChibiOS objects factory structures and macros ........ 361
- **chjobs.h**: Jobs Queues structures and macros ............................ 364
- **chlib.h**: ChibiOS/LIB main include file .................................... 366
- **chmboxes.c**: Mailboxes code ..................................................... 367
- **chmboxes.h**: Mailboxes macros and structures ............................ 367
- **chmemcore.c**: Core memory manager code .................................. 369
- **chmemcore.h**: Core memory manager macros and structures ........... 369
- **chmemheaps.c**: Memory heaps code .......................................... 370
- **chmemheaps.h**: Memory heaps macros and structures ................... 371
<table>
<thead>
<tr>
<th>File Name</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>chmempools.c</td>
<td>Memory Pools code</td>
<td>372</td>
</tr>
<tr>
<td>chmempools.h</td>
<td>Memory Pools macros and structures</td>
<td>373</td>
</tr>
<tr>
<td>chmsg.c</td>
<td>Nil RTOS synchronous messages source file</td>
<td>374</td>
</tr>
<tr>
<td>chmsg.h</td>
<td>Nil RTOS synchronous messages header file</td>
<td>375</td>
</tr>
<tr>
<td>chobjcaches.c</td>
<td>Objects Caches code</td>
<td>375</td>
</tr>
<tr>
<td>chobjcaches.h</td>
<td>Objects Caches macros and structures</td>
<td>377</td>
</tr>
<tr>
<td>chobjfifos.h</td>
<td>Objects FIFO structures and macros</td>
<td>378</td>
</tr>
<tr>
<td>chpipes.c</td>
<td>Pipes code</td>
<td>379</td>
</tr>
<tr>
<td>chpipes.h</td>
<td>Pipes macros and structures</td>
<td>380</td>
</tr>
<tr>
<td>chsem.c</td>
<td>Nil RTOS semaphores source file</td>
<td>381</td>
</tr>
<tr>
<td>chsem.h</td>
<td>Nil RTOS semaphores header file</td>
<td>382</td>
</tr>
</tbody>
</table>
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
- Base API
- Semaphores
- Events
- Synchronous Messages
6.2 Configuration

6.2.1 Detailed Description

Kernel related settings and hooks.

Kernel parameters and options

- `#define CH_CFG_MAX_THREADS 4`
  Maximum number of user threads in the application.
- `#define CH_CFG_AUTOSTART_THREADS TRUE`
  Auto starts threads when `chSysInit()` is invoked.

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_WAITEXIT TRUE`
  Threads synchronization APIs.
- `#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_MESSAGES TRUE`
  Synchronous Messages APIs.

OSLIB options

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

Objects factory options

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

Debug options

• \#define CH_DBG_STATISTICS FALSE
   Debug option, kernel statistics.
• \#define CH_DBG_SYSTEM_STATE_CHECK TRUE
   Debug option, system state check.
• \#define CH_DBG_ENABLE_CHECKS TRUE
   Debug option, parameters checks.
• \#define CH_DBG_ENABLE_ASSERTS TRUE
   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_THREAD_EXIT_HOOK(tp) {}`
  Threads finalization 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 CH_CFG_MAX_THREADS

`#define CH_CFG_MAX_THREADS 4`

Maximum number of user threads in the application.

**Note**

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

Set this value to be exactly equal to the number of threads you will use or you would be wasting RAM and cycles.

This values also defines the number of available priorities (0..CH_CFG_MAX_THREADS-1).

6.2.2.2 CH_CFG_AUTOSTART_THREADS

`#define CH_CFG_AUTOSTART_THREADS TRUE`

Auto starts threads when `chSysInit()` is invoked.
6.2.2.3 CH_CFG_ST_RESOLUTION

#define CH_CFG_ST_RESOLUTION 32

System time counter resolution.

**Note**
- Allowed values are 16 or 32 bits.

6.2.2.4 CH_CFG_ST_FREQUENCY

#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.

6.2.2.5 CH_CFG_ST_TIMEDELTA

#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.

6.2.2.6 CH_CFG_USE_WAITEXIT

#define CH_CFG_USE_WAITEXIT TRUE

Threads synchronization APIs.

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

**Note**
- The default is TRUE.
6.2.2.7 CH_CFG_USE_SEMAPHORES

#define CH_CFG_USE_SEMAPHORES TRUE

Semaphores APIs.

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

Note

The default is TRUE.

6.2.2.8 CH_CFG_USE_MUTEXES

#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.

6.2.2.9 CH_CFG_USE_EVENTS

#define CH_CFG_USE_EVENTS TRUE

Events Flags APIs.

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

Note

The default is TRUE.

6.2.2.10 CH_CFG_USE_MESSAGES

#define CH_CFG_USE_MESSAGES TRUE

Synchronous Messages APIs.

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

Note

The default is TRUE.
6.2.2.11 CH_CFG_USE_MAILBOXES

#define CH_CFG_USE_MAILBOXES TRUE

Mailboxes APIs.

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

Note
The default is TRUE.
Requires CH_CFG_USE_SEMAPHORES.

6.2.2.12 CH_CFG_USE_MEMCORE

#define CH_CFG_USE_MEMCORE TRUE

Core Memory Manager APIs.

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

Note
The default is TRUE.

6.2.2.13 CH_CFG_MEMCORE_SIZE

#define CH_CFG_MEMCORE_SIZE 0

Managed RAM size.

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

Note
In order to let the OS manage the whole RAM the linker script must provide the heap_base and heap_end symbols.
Requires CH_CFG_USE_MEMCORE.
6.2.2.14 CH_CFG_USE_HEAP

#define CH_CFG_USE_HEAP TRUE

Heap Allocator APIs.

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

Note

The default is TRUE.

6.2.2.15 CH_CFG_USE_MEMPOOLS

#define CH_CFG_USE_MEMPOOLS TRUE

Memory Pools Allocator APIs.

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

Note

The default is TRUE.

6.2.2.16 CH_CFG_USE_OBJ_FIFOS

#define CH_CFG_USE_OBJ_FIFOS TRUE

Objects FIFOs APIs.

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

Note

The default is TRUE.

6.2.2.17 CH_CFG_USE_PIPES

#define CH_CFG_USE_PIPES TRUE

Pipes APIs.

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

Note

The default is TRUE.
6.2.2.18 \textbf{CH\_CFG\_USE\_OBJ\_CACHES}

\texttt{#define CH\_CFG\_USE\_OBJ\_CACHES TRUE}

Objects Caches APIs.

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

\textbf{Note}

The default is \texttt{TRUE}.

6.2.2.19 \textbf{CH\_CFG\_USE\_DELEGATES}

\texttt{#define CH\_CFG\_USE\_DELEGATES TRUE}

Delegate threads APIs.

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

\textbf{Note}

The default is \texttt{TRUE}.

6.2.2.20 \textbf{CH\_CFG\_USE\_JOBS}

\texttt{#define CH\_CFG\_USE\_JOBS TRUE}

Jobs Queues APIs.

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

\textbf{Note}

The default is \texttt{TRUE}.

6.2.2.21 \textbf{CH\_CFG\_USE\_FACTORY}

\texttt{#define CH\_CFG\_USE\_FACTORY TRUE}

Objects Factory APIs.

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

\textbf{Note}

The default is \texttt{FALSE}.
6.2.2.22 CH_CFG_FACTORY_MAX_NAMES_LENGTH

#define CH_CFG_FACTORY_MAX_NAMES_LENGTH 8

Maximum length for object names.

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

6.2.2.23 CH_CFG_FACTORY_OBJECTS_REGISTRY

#define CH_CFG_FACTORY_OBJECTS_REGISTRY TRUE

Enables the registry of generic objects.

6.2.2.24 CH_CFG_FACTORY_GENERIC_BUFFERS

#define CH_CFG_FACTORY_GENERIC_BUFFERS TRUE

Enables factory for generic buffers.

6.2.2.25 CH_CFG_FACTORY_SEMAPHORES

#define CH_CFG_FACTORY_SEMAPHORES TRUE

Enables factory for semaphores.

6.2.2.26 CH_CFG_FACTORY_MAILBOXES

#define CH_CFG_FACTORY_MAILBOXES TRUE

Enables factory for mailboxes.

6.2.2.27 CH_CFG_FACTORY_OBJ_FIFOS

#define CH_CFG_FACTORY_OBJ_FIFOS TRUE

Enables factory for objects FIFOs.
6.2.2.28 CH_CFG_FACTORY_PIPES

#define CH_CFG_FACTORY_PIPES TRUE

Enables factory for Pipes.

6.2.2.29 CH_DBG_STATISTICS

#define CH_DBG_STATISTICS FALSE

Debug option, kernel statistics.

Note

Feature not currently implemented.
The default is FALSE.

6.2.2.30 CH_DBG_SYSTEM_STATE_CHECK

#define CH_DBG_SYSTEM_STATE_CHECK TRUE

Debug option, system state check.

Note

The default is FALSE.

6.2.2.31 CH_DBG_ENABLE_CHECKS

#define CH_DBG_ENABLE_CHECKS TRUE

Debug option, parameters checks.

Note

The default is FALSE.
6.2.2.32 CH_DBG_ENABLE_ASSERTS

#define CH_DBG_ENABLE_ASSERTS TRUE

System assertions.

Note

The default is FALSE.

6.2.2.33 CH_DBG_ENABLE_STACK_CHECK

#define CH_DBG_ENABLE_STACK_CHECK TRUE

Stack check.

Note

The default is FALSE.

6.2.2.34 CH_CFG_SYSTEM_INIT_HOOK

#define CH_CFG_SYSTEM_INIT_HOOK( )

Value:

{
    // System initialization hook.
}

6.2.2.35 CH_CFG_THREAD_EXT_FIELDS

#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.

6.2.2.36 CH_CFG_THREAD_EXT_INIT_HOOK

#define CH_CFG_THREAD_EXT_INIT_HOOK( tr )

Value:

{
    /* Add custom threads initialization code here.*/
    // Threads initialization hook.
}
6.2.2.37 CH_CFG_THREAD_EXIT_HOOK

#define CH_CFG_THREAD_EXIT_HOOK(t)

Threads finalization hook.
User finalization code added to the chThdExit() API.

6.2.2.38 CH_CFG_IDLE_ENTER_HOOK

#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.

6.2.2.39 CH_CFG_IDLE_LEAVE_HOOK

#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.

6.2.2.40 CH_CFG_SYSTEM_HALT_HOOK

#define CH_CFG_SYSTEM_HALT_HOOK(reason)

Value:
[
 ]

System halt hook.
6.3 Base API

6.3.1 Detailed Description

Macros

- #define __CHIBIOS_NIL__
  ChibiOS/NIL identification macro.
- #define CH_KERNEL_STABLE 0
  Stable release flag.
- #define THD_IDLE_BASE (&__main_thread_stack_base__)
- #define __CH_STRINGIFY(a) #a
  Utility to make the parameter a quoted string.
- #define THD_WORKING_AREA_END(wa) ((wa) + ((sizeof wa) / sizeof (stkalign_t)))
  Returns the top address of a working area.

ChibiOS/NIL version identification

- #define CH_KERNEL_VERSION "4.1.0"
  Kernel version string.
- #define CH_KERNEL_MAJOR 4
  Kernel version major number.
- #define CH_KERNEL_MINOR 1
  Kernel version minor number.
- #define CH_KERNEL_PATCH 0
  Kernel version patch number.

Constants for configuration options

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

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_WSTART (tstate_t)0
  Thread not yet started or terminated.
- \#define NIL_STATE_READY (tstate_t)1
  Thread ready or executing.
- \#define NIL_STATE_SLEEPING (tstate_t)2
  Thread sleeping.
- \#define NIL_STATE_SUSPENDED (tstate_t)3
  Thread suspended.
- \#define NIL_STATE_WEXIT (tstate_t)4
  Waiting a thread.
- \#define NIL_STATE_WTQUEUE (tstate_t)5
  On queue or semaph.
- \#define NIL_STATE_WTREVT (tstate_t)6
  Waiting for events.
- \#define NIL_STATE_WTANDEVT (tstate_t)7
  Waiting for events.
- \#define NIL_STATE_SNDSMQQ (tstate_t)8
  Sending a message, in queue.
- \#define NIL_STATE_WTMSG (tstate_t)10
  Waiting for a message.
- \#define NIL_STATE_FINAL (tstate_t)11
  Thread terminated.

ChibiOS/NIL
RT options not existing in NIL

- `#define CH_CFG_USE_REGISTRY FALSE`

Kernel types

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

Threads tables definition macros

- `#define THD_TABLE_BEGIN const thread_descriptor_t nil_thd_configs[] = {
  Start of user threads table.
  #define THD_TABLE_THREAD(_prio, _name, _wap, _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.

ISR 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_QUEUEDECL(name) threads_queue_t name = __THREADS_QUEUE_DATA(name)`
  Static threads queue object initializer.
Macro Functions

- `#define chSysGetRealtimeCounterX(rctnt_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 chSchGoSleepS(newState) chSchGoSleepTimeoutS(newState, TIME_INFINITE)`
  
  Puts the current thread to sleep into the specified state.

- `#define chSchWakeupS(ntp, msg)`
  
  Wakes up a thread.

- `#define chSchIsRescRequiredI() ((bool)(nil.current != nil.next))`
  
  Evaluates if a reschedule is required.

- `#define chThdGetSelfX()`
  
  Returns a pointer to the current thread_t.

- `#define chThdGetPriorityX(void) (tprio_t)(nil.current - &nil.threads[0])`
  
  Returns the current thread priority.

- `#define chThdResumeS(trp, msg)`
  
  Wakes up a thread waiting on a thread reference object.

- `#define chThdSleepSeconds(secs) chThdSleep(SLEEP_S2I(secs))`
  
  Delays the invoking thread for the specified number of seconds.

- `#define chThdSleepMilliseconds(msecs) chThdSleep(SLEEP_MS2I(msecs))`
  
  Delays the invoking thread for the specified number of milliseconds.

- `#define chThdSleepMicroseconds(usecs) chThdSleep(SLEEP_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(tpq) ((tpq)->cnt = (cnt_t)0)`
  
  Initializes a threads queue object.

- `#define chThdQueueIsEmptyI(tpq) ((bool)(tpq)->cnt == (cnt_t)0)`
  
  Evaluates to true if the specified queue is empty.

- `#define chVTGetSystemTimeX()`
  
  Current system time.

- `#define chVTTimeElapsedSinceX(start) chTimeDiffX((start), chVTGetSystemTimeX())`
  
  Returns the elapsed time since the specified start time.

- `#define chVTIsSystemTimeWithinX(start, end) chTimeIsInRangeX(chVTGetSystemTimeX(), start, end)`
  
  Checks if the current system time is within the specified time window.

- `#define chTimeAddX(systime, interval) ((systime_t)(systime) + (systime_t)(interval))`
6.3 Base API

**Base API**

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

- `#define chTimeDiffX(start, end) ((sysinterval_t)((systime_t)((systime_t)(end) - (systime_t)(start))))`

Subtracts two system times returning an interval.

- `#define chDbgCheck(c)`: Function parameters check.

- `#define chDbgAssert(c, r)`: Condition assertion.

### Typedefs

- `typedef uint32_t systime_t`: Type of system time.
- `typedef uint32_t sysinterval_t`: Type of time interval.
- `typedef uint64_t time_conv_t`: Type of time conversion variable.
- `typedef struct nil_os_instance os_instance_t`: Type of a structure representing the system.
- `typedef void(*tfunc_t)(void *)`: Thread function.
- `typedef struct nil_thread_descriptor thread_descriptor_t`: Type of a thread descriptor.
- `typedef struct nil_thread thread_t`: Type of a structure representing a thread.
- `typedef thread_t *thread_reference_t`: Type of a thread reference.
- `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.

### Data Structures

- `struct nil_threads_queue`: Structure representing a queue of threads.
- `struct nil_thread_descriptor`: Structure representing a thread descriptor.
- `struct nil_thread`: Structure representing a thread.
- `struct nil_os_instance`: System data structure.
Functions

- 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.
- 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 chSchDoPreemption (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.

- **bool chTimesInRangeX (systime_t time, systime_t start, systime_t end)**
  Checks if the specified time is within the specified time range.

- **thread_t * chThdCreateI (const thread_descriptor_t *tdp)**
  Creates a new thread into a static memory area.

- **thread_t * chThdCreate (const thread_descriptor_t *tdp)**
  Creates a new thread into a static memory area.

- **void chThdExit (msg_t msg)**
  Terminates the current thread.

- **msg_t chThdWait (thread_t *tp)**
  Blocks the execution of the invoking thread until the specified thread terminates then the exit code is returned.

- **msg_t 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 (systime_t abstime)**
  Suspends the invoking thread until the system time arrives to the specified value.

- **msg_t chThdEnqueueTimeoutS (threads_queue_t *tqp, sysinterval_t timeout)**
  Enqueues the caller thread on a threads queue object.

- **void chThdDoDequeueNextI (threads_queue_t *tqp, msg_t msg)**
  Dequeues and wakes up one thread from the threads queue object.

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

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

### Variables

- **os_instance_t nil**
  System data structures.

### 6.3.2 Macro Definition Documentation

#### 6.3.2.1 __CHIBIOS_NIL__

```c
#define __CHIBIOS_NIL__
```

ChibiOS/NIL identification macro.
6.3.2.2 CH_KERNEL_STABLE

#define CH_KERNEL_STABLE 0

Stable release flag.

6.3.2.3 CH_KERNEL_VERSION

#define CH_KERNEL_VERSION "4.1.0"

Kernel version string.

6.3.2.4 CH_KERNEL_MAJOR

#define CH_KERNEL_MAJOR 4

Kernel version major number.

6.3.2.5 CH_KERNEL_MINOR

#define CH_KERNEL_MINOR 1

Kernel version minor number.

6.3.2.6 CH_KERNEL_PATCH

#define CH_KERNEL_PATCH 0

Kernel version patch number.

6.3.2.7 FALSE

#define FALSE 0

Generic 'false' preprocessor boolean constant.

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

#define TRUE 1

Generic 'true' preprocessor boolean constant.

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

6.3.2.9 MSG_OK

#define MSG_OK (msg_t)0

OK wakeup message.

6.3.2.10 MSG_TIMEOUT

#define MSG_TIMEOUT (msg_t)-1

Wake-up caused by a timeout condition.

6.3.2.11 MSG_RESET

#define MSG_RESET (msg_t)-2

Wake-up caused by a reset condition.

6.3.2.12 TIME_IMMEDIATE

#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 TIME_INFINITE

#define TIME_INFINITE ((sysinterval_t)0)

Infinite time specification for all functions with a timeout specification.

6.3.2.14 TIME_MAX_INTERVAL

#define TIME_MAX_INTERVAL ((sysinterval_t)-2)

Maximum interval constant usable as timeout.

6.3.2.15 TIME_MAX_SYSTIME

#define TIME_MAX_SYSTIME ((systime_t)-1)

Maximum system of system time before it wraps.

6.3.2.16 NIL_STATE_WTSTART

#define NIL_STATE_WTSTART (tstate_t)0

Thread not yet started or terminated.

6.3.2.17 NIL_STATE_READY

#define NIL_STATE_READY (tstate_t)1

Thread ready or executing.

6.3.2.18 NIL_STATE_SLEEPING

#define NIL_STATE_SLEEPING (tstate_t)2

Thread sleeping.
6.3.2.19  **NIL_STATE_SUSPENDED**

#define NIL_STATE_SUSPENDED (tstate_t)3  
Thread suspended.

6.3.2.20  **NIL_STATE_WTEXIT**

#define NIL_STATE_WTEXIT (tstate_t)4  
Waiting a thread.

6.3.2.21  **NIL_STATE_WTQUEUE**

#define NIL_STATE_WTQUEUE (tstate_t)5  
On queue or semaph.

6.3.2.22  **NIL_STATE_WTOREVT**

#define NIL_STATE_WTOREVT (tstate_t)6  
Waiting for events.

6.3.2.23  **NIL_STATE_WTANDEVT**

#define NIL_STATE_WTANDEVT (tstate_t)7  
Waiting for events.

6.3.2.24  **NIL_STATE_SNDMSGQ**

#define NIL_STATE_SNDMSGQ (tstate_t)8  
Sending a message, in queue.
6.3.2.25  NIL_STATE_WTMNG

#define NIL_STATE_WTMNG (tstate_t)10

Waiting for a message.

6.3.2.26  NIL_STATE_FINAL

#define NIL_STATE_FINAL (tstate_t)11

Thread terminated.

6.3.2.27  THD_IDLE_BASE

#define THD_IDLE_BASE (&__main_thread_stack_base__)

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

6.3.2.28  __CH_STRINGIFY

#define __CH_STRINGIFY(
        a
    ) #a

Utility to make the parameter a quoted string.

6.3.2.29  THD_TABLE_BEGIN

#define THD_TABLE_BEGIN const thread_descriptor_t nil_thd_configs[] = {

Start of user threads table.
6.3.2.30 THD_TABLE_THREAD

#define THD_TABLE_THREAD(_prio, _name, _wap, _funcp, _arg)

Value:
{
    .name = (_name),
    .wbase = (_wap),
    .wend = THD_WORKING_AREA_END(_wap),
    .prio = (_prio),
    .funcp = (_funcp),
    .arg = (_arg)
}

Entry of user threads table.

6.3.2.31 THD_TABLE_END

#define THD_TABLE_END

Value:
{
    .name = "idle",
    .wbase = THD_IDLE_BASE,
    .wend = THD_IDLE_END,
    .prio = CH_CFG_MAX_THREADS,
    .funcp = NULL,
    .arg = NULL
}

End of user threads table.

6.3.2.32 MEM_ALIGN_MASK

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

Alignment mask constant.

Parameters

in a alignment, must be a power of two

6.3.2.33 MEM_ALIGN_PREV

#define MEM_ALIGN_PREV(}
Aligns to the previous aligned memory address.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>p</th>
<th>variable to be aligned</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>a</td>
<td>alignment, must be a power of two</td>
</tr>
</tbody>
</table>

### 6.3.2.34 MEM_ALIGN_NEXT

```c
#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

<table>
<thead>
<tr>
<th>in</th>
<th>p</th>
<th>variable to be aligned</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>a</td>
<td>alignment, must be a power of two</td>
</tr>
</tbody>
</table>

### 6.3.2.35 MEM_IS_ALIGNED

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

Returns whatever a pointer or memory size is aligned.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>p</th>
<th>variable to be aligned</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>a</td>
<td>alignment, must be a power of two</td>
</tr>
</tbody>
</table>

### 6.3.2.36 MEM_IS_VALID_ALIGNMENT

```c
#define MEM_IS_VALID_ALIGNMENT(a)
```

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

Returns 1 if the alignment is valid; otherwise, 0.
Returns whatever a constant is a valid alignment.

Valid alignments are powers of two.

Parameters

| in | a | alignment to be checked, must be a constant |

### 6.3.2.37 THD_WORKING_AREA_SIZE

```c
#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 | n | 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.

### 6.3.2.38 THD_WORKING_AREA

```c
#define THD_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 | s | the name to be assigned to the stack array |
| in | n | 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.

6.3.2.39 THD_WORKING_AREA_END

#define THD_WORKING_AREA_END(wa) ((wa) + ((sizeof wa) / sizeof (stkalign_t)))

Returns the top address of a working area.

Note
The parameter is assumed to be an array of stkalign_t. The macros is invalid for anything else.

Parameters

| in | wa | working area array |

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

6.3.2.40 THD_FUNCTION

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

Thread declaration macro.

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

6.3.2.41 CH_IRQ_IS_VALID_PRIORITY

#define CH_IRQ_IS_VALID_PRIORITY(prio) PORT_IRQ_IS_VALID_PRIORITY(prio)

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

#define CH_IRQ_IS_VALID_KERNEL_PRIORITY(
prio ) PORT_IRQ_IS_VALID_KERNEL_PRIORITY(prio)

Priority level validation macro.

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

Parameters

in prio the priority level

Returns

Priority range result.

Return values

false if the priority is invalid or if the architecture does not support priorities.
true if the priority is valid.

6.3.2.43 CH_IRQ_PROLOGUE

#define CH_IRQ_PROLOGUE()

Value:

PORT_IRQ_PROLOGUE();
__dbg_check_enter_isr();

IRQ handler enter code.
Note

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

Function Class:

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

6.3.2.44 CH_IRQ_EPILOGUE

#define CH_IRQ_EPILOGUE( )

Value:
\__dbg_check_leave_isr()\
PORT_IRQ_EPILOGUE()

IRQ handler exit code.

Note

Usually IRQ handlers function are also declared naked.

Function Class:

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

6.3.2.45 CH_IRQ_HANDLER

#define CH_IRQ_HANDLER(
    id ) PORT_IRQ_HANDLER(id)

Standard normal IRQ handler declaration.

Note

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

Function Class:

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

#define CH_FAST_IRQ_HANDLER(id) PORT_FAST_IRQ_HANDLER(id)

Standard fast IRQ handler declaration.

Note

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

Function Class:

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

6.3.2.47 TIME_S2I

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

Seconds to time interval.

Converts from seconds to system ticks number.

Note

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

Parameters

| in | secs | number of seconds |

Returns

The number of ticks.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.
### 6.3.2.48 TIME_MS2I

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

**Value:**

\[
\text{sysinterval_t} = \left( \frac{\text{sysinterval_t}\left(\text{time_conv_t}(\text{msecs}) \times \text{CH_CFG_ST_FREQUENCY}\right) + 999}{1000} \right)
\]

Milliseconds to time interval.

Converts from milliseconds to system ticks number.

**Note**

The result is rounded upward to the next tick boundary.

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

**Parameters**

| in | msecs | number of milliseconds |

**Returns**

The number of ticks.

**Function Class:**

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

### 6.3.2.49 TIME_US2I

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

**Value:**

\[
\text{sysinterval_t} = \left( \frac{\text{sysinterval_t}\left(\text{time_conv_t}(\text{usecs}) \times \text{CH_CFG_ST_FREQUENCY}\right) + 999999}{1000000} \right)
\]

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 usecs number of microseconds

Returns

The number of ticks.

Function Class:

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

6.3.2.50 TIME_I2S

#define TIME_I2S(
    interval)

Value:

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

Time interval to seconds.

Converts from system ticks number to seconds.

Note

The result is rounded up to the next second boundary.

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

Parameters

in interval interval in ticks

Returns

The number of seconds.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.
6.3.2.51 TIME_I2MS

#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  interval  interval in ticks

Returns
The number of milliseconds.

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

6.3.2.52 TIME_I2US

#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 | interval | interval in ticks |

Returns

The number of microseconds.

Function Class:

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

6.3.2.53 __THREADS_QUEUE_DATA

#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 | name | the name of the threads queue variable |

6.3.2.54 THREADS_QUEUE_DECL

#define THREADS_QUEUE_DECL(
    name ) threads_queue_t name = __THREADS_QUEUE_DATA(name)

Static threads queue object initializer.

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

Parameters

| in | name | the name of the threads queue variable |

6.3.2.55 chSysGetRealtimeCounterX

#define chSysGetRealtimeCounterX( ) (rtcnt_t)port_rt_get_counter_value()

Returns the current value of the system real time counter.
Note
This function is only available if the port layer supports the option 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.

6.3.2.56 chSysDisable

#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.

6.3.2.57 chSysSuspend

#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.
6.3.2.58 chSysEnable

#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.

6.3.2.59 chSysLock

#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.

6.3.2.60 chSysUnlock

#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.
6.3.2.61 chSysLockFromISR

#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.

6.3.2.62 chSysUnlockFromISR

#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.

6.3.2.63 chSchGoSleepS

#define chSchGoSleepS( newstate ) chSchGoSleepTimeoutS(newstate, TIME_INFINITE)

Puts the current thread to sleep into the specified state.
6.3 Base API

Parameters:

| in    | newstate | the new thread state or a semaphore pointer |

Returns:

The wakeup message.

Function Class:

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

6.3.2.64 chSchWakeupS

#define chSchWakeupS(
   ntp,
   msg )

Value:

do {
   chSchReadyI(ntp, msg);
   chSchRescheduleS();
} while (false)

Wakes up a thread.

Parameters:

| in    | ntp     | the thread to be made ready |
| in    | msg     | the wakeup message          |

Function Class:

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

6.3.2.65 chSchIsRescRequiredI

#define chSchIsRescRequiredI( ) ((bool)(nil.current != nil.next))

Evaluates if a reschedule is required.

Return values:

| true  | if there is a thread that must go in running state immediately. |
| false | if preemption is not required.     |
Function Class:
   This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

### 6.3.2.66 chThdGetSelfX

```c
#define chThdGetSelfX() nil.current
```

Returns a pointer to the current `thread_t`.

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

### 6.3.2.67 chThdGetPriorityX

```c
#define chThdGetPriorityX( void ) (tprio_t)(nil.current - &nil.threads[0])
```

Returns the current thread priority.

**Note**
   Can be invoked in any context.

**Returns**
   The current thread priority.

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

### 6.3.2.68 chThdResumeS

```c
#define chThdResumeS( trp, msg )
```

**Value:**

```c
do { 
    chThdResumeI(trp, msg); 
    chSchRescheduleS(); 
} while (false)
```

Wakes up a thread waiting on a thread reference object.

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

<table>
<thead>
<tr>
<th>in</th>
<th>trp</th>
<th>a pointer to a thread reference object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>msg</td>
<td>the message code</td>
</tr>
</tbody>
</table>

Function Class:

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

### 6.3.2.69 chThdSleepSeconds

```c
#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  | secs | 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 chThdSleepMilliseconds

```c
#define chThdSleepMilliseconds(
    msecs ) 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   | msecs | 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 chThdSleepMicroseconds

#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   | usecs | 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 chThdSleepS

#define chThdSleepS(
    timeout ) (void) ch3chGoSleepTimeoutS(NIL_STATE_SLEEPING, timeout)

Suspends the invoking thread for the specified time.

Parameters

| in   | timeout | 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 chThdSleepUntilS

```c
#define chThdSleepUntilS(
    abstime )
```

**Value:**

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

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

**Parameters**

- `abstime` 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 chThdQueueObjectInit

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

Initializes a threads queue object.

**Parameters**

- `tqp` 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 chThdQueueIsEmptyI

```c
#define chThdQueueIsEmptyI(
    tqp ) ((bool)(tqp->cnt >= (cnt_t)0))
```

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

**Parameters**

- `tqp` pointer to the threads queue object
Returns

The queue status.

Return values

<table>
<thead>
<tr>
<th>false</th>
<th>if the queue is not empty.</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>if the queue is empty.</td>
</tr>
</tbody>
</table>

Function 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 chVTGetSystemTimeX

#define chVTGetSystemTimeX( ) (nil.systime)

Current system time.

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

Note

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

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

Returns

The system time in ticks.

Function Class:

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

6.3.2.77 chVTTimeElapsedSinceX

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

Returns the elapsed time since the specified start time.
6.3 Base API

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>start</th>
<th>start time</th>
</tr>
</thead>
</table>

Returns

The elapsed time.

Function Class:

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

6.3.2.78 chVTIsSystemTimeWithinX

```c
#define chVTIsSystemTimeWithinX(
    start,
    end
) chTimeIsInRangeX(chVTGetSystemTimeX(), start, end)
```

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

**Note**

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

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>start</th>
<th>the start of the time window (inclusive)</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>end</td>
<td>the end of the time window (non inclusive)</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>true</th>
<th>current time within the specified time window.</th>
</tr>
</thead>
<tbody>
<tr>
<td>false</td>
<td>current time not within the specified time window.</td>
</tr>
</tbody>
</table>

Function Class:

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

6.3.2.79 chTimeAddX

```c
#define chTimeAddX(
    systime,
    interval
) {{systime_t}(systime) + (systime_t)(interval)}
```

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

```c
#define chTimeDiffX(
    start,
    end ) ((sysinterval_t)((systime_t)((systime_t)(end) - (systime_t)(start))))
```

Subtracts two system times returning an interval.

### Parameters

<table>
<thead>
<tr>
<th>in</th>
<th><code>systime</code></th>
<th>base system time</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td><code>interval</code></td>
<td>interval to be added</td>
</tr>
</tbody>
</table>

### Returns

The new system time.

### Function Class:

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

6.3.2.81  chDbgCheck

```c
#define chDbgCheck(
    c )
```

Value:

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

Function parameters check.

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

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

Parameters

| in | c | the condition to be verified to be true |

Function Class:

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

6.3.2.82 chDbgAssert

#define chDbgAssert(
  c,
  r )

Value:

\do |
  
  /*lint -save -e506 -e774 [2.1, 14.3] Can be a constant by design.*)*/
  
  if (CH_DBG_ENABLE_ASSERTS != FALSE) {
    if (!(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 | c | the condition to be verified to be true |
| in | r | 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 rtcnt_t

typedef port_rtcnt_t rtcnt_t

Realtime counter.

6.3.3.2 syssts_t

typedef port_syssts_t syssts_t

System status word.

6.3.3.3 stkalign_t

typedef port_stkalign_t stkalign_t

Stack alignment type.

6.3.3.4 tstate_t

typedef uint8_t tstate_t

Thread state.

6.3.3.5 tprio_t

typedef uint32_t tprio_t

Thread priority.

6.3.3.6 msg_t

typedef int32_t msg_t

Inter-thread message.

6.3.3.7 eventid_t

typedef int32_t eventid_t

Numeric event identifier.
6.3 Base API

6.3.3.8 eventmask_t

typedef uint32_t eventmask_t

Mask of event identifiers.

6.3.3.9 eventflags_t

typedef uint32_t eventflags_t

Mask of event flags.

6.3.3.10 cnt_t

typedef int32_t cnt_t

Generic signed counter.

6.3.3.11 ucnt_t

typedef uint32_t ucnt_t

Generic unsigned counter.

6.3.3.12 systime_t

typedef uint32_t systime_t

Type of system time.

Note

It is selectable in configuration between 16 or 32 bits.

6.3.3.13 sysinterval_t

typedef uint32_t sysinterval_t

Type of time interval.

Note

It is selectable in configuration between 16 or 32 bits.
6.3.3.14 time_conv_t

typedef uint64_t time_conv_t

Type of time conversion variable.

Note

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

6.3.3.15 os_instance_t

typedef struct nil_os_instance os_instance_t

Type of a structure representing the system.

6.3.3.16 tfunc_t

typedef void(* tfunc_t) (void *p)

Thread function.

6.3.3.17 thread_descriptor_t

typedef struct nil_thread_descriptor thread_descriptor_t

Type of a thread descriptor.

6.3.3.18 thread_t

typedef struct nil_thread thread_t

Type of a structure representing a thread.

Note

It is required as an early definition.
6.3.3.19 thread_reference_t

typedef thread_t* thread_reference_t

Type of a thread reference.

6.3.3.20 threads_queue_t

typedef struct nil_threads_queue threads_queue_t

Type of a queue of threads.

6.3.3.21 semaphore_t

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.

6.3.4 Function Documentation

6.3.4.1 nil_find_thread()

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.

Parameters

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>in</strong></td>
<td><strong>state</strong></td>
<td>thread state</td>
</tr>
<tr>
<td><strong>in</strong></td>
<td><strong>p</strong></td>
<td>object pointer</td>
</tr>
</tbody>
</table>

Returns

The pointer to the found thread.
Return values

| NULL | if the thread is not found |

Function Class:

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

### 6.3.4.2 nil_ready_all()

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.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>p</th>
<th>object pointer</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>cnt</td>
<td>number of threads to be readied as a negative number, non negative numbers are ignored</td>
</tr>
<tr>
<td>in</td>
<td>msg</td>
<td>the wakeup message</td>
</tr>
</tbody>
</table>

**Returns**

The number of readied threads.

Function Class:

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

Here is the call graph for this function:

```
| nil_ready_all | chSchReadyI | chDbgCheckClassI |
```
6.3.4.3  __dbg_check_disable()

void __dbg_check_disable (  
    void )  

Guard code for chSysDisable().

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

Here is the call graph for this function:

![Call graph for __dbg_check_disable()](image)

6.3.4.4  __dbg_check_suspend()

void __dbg_check_suspend (  
    void )  

Guard code for chSysSuspend().

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

Here is the call graph for this function:

![Call graph for __dbg_check_suspend()](image)
6.3.4.5 __dbg_check_enable()

```c
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:

![Call Graph for __dbg_check_enable()](call_graph.png)

6.3.4.6 __dbg_check_lock()

```c
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:

![Call Graph for __dbg_check_lock()](call_graph.png)
6.3.4.7 __dbg_check_unlock()

```c
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:

![Call Graph for __dbg_check_unlock()](image)

6.3.4.8 __dbg_check_lock_from_isr()

```c
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:

![Call Graph for __dbg_check_lock_from_isr()](image)
6.3.4.9  __dbg_check_unlock_from_isr()

void __dbg_check_unlock_from_isr (  
    void  )

Guard code for chSysUnlockFromIsr().

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

Here is the call graph for this function:

6.3.4.10  __dbg_check_enter_isr()

void __dbg_check_enter_isr (  
    void  )

Guard code for CH_IRQ_PROLOGUE().

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

Here is the call graph for this function:
6.3.4.11  __dbg_check_leave_isr()

void __dbg_check_leave_isr (
    void)

Guard code for CH_IRQ_EPILOGUE().

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

Here is the call graph for this function:

```
__dbg_check_leave_isr  chSysHalt
```

6.3.4.12  chDbgCheckClassI()

void chDbgCheckClassI (
    void)

I-class functions context check.

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

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

Here is the call graph for this function:

```
chDbgCheckClassI  chSysHalt
```
6.3.4.13 chDbgCheckClassS()

```c
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:

```
chDbgCheckClassS -> chSysHalt
```

6.3.4.14 chSysInit()

```c
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.
6.3 Base API

Function Class:
Special function, this function has special requirements see the notes.

Here is the call graph for this function:

6.3.4.15 chSysHalt()

```c
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 reason pointer to an error string
```

Function Class:
Special function, this function has special requirements see the notes.
6.3.4.16 chSysTimerHandlerI()

```c
void chSysTimerHandlerI ( void )
```

Time management handler.

**Note**

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

**Function Class:**

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

Here is the call graph for this function:

![Call Graph](chSysTimerHandlerI chDbgCheckClassI chSysHalt)

6.3.4.17 chSysUnconditionalLock()

```c
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  chSysUnconditionalUnlock()

```c
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  chSysGetStatusAndLockX()

```c
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  chSysRestoreStatusX()

```c
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:

```
chSysRestoreStatusX  ->  chSchRescheduleS  ->  chDbgCheckClassS  ->  chSchDoPreemption
```

### 6.3.4.21 chSysIsCounterWithinX()

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

Realtime window test.

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

**Note**

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

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

| true | current time within the specified time window. |
| false | 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 chSysPolledDelayX()

```c
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  | cycles | 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:

```
chSysPolledDelayX → chSysIsCounterWithinX
```

### 6.3.4.23 chSchReadyI()

```c
thread_t * chSchReadyI ( 
    thread_t * tp, 
    msg_t msg )
```

Makes the specified thread ready for execution.
### Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>tp</th>
<th>pointer to the thread_t object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>msg</td>
<td>the wakeup message</td>
</tr>
</tbody>
</table>

### Returns

The same reference passed as parameter.

Here is the call graph for this function:

![Call Graph](image)

#### 6.3.4.24 chSchIsPreemptionRequired()

```c
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

<table>
<thead>
<tr>
<th>true</th>
<th>if there is a thread that must go in running state immediately.</th>
</tr>
</thead>
<tbody>
<tr>
<td>false</td>
<td>if preemption is not required.</td>
</tr>
</tbody>
</table>

### Function Class:

Special function, this function has special requirements see the notes.
6.3.4.25  chSchDoPreemption()

```
void chSchDoPreemption {
    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  chSchRescheduleS()

```
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:

![Call Graph]

6.3.4.27  chSchGoSleepTimeoutS()

```
msg_t chSchGoSleepTimeoutS {
    tstate_t newstate,
    sysinterval_t timeout 
}
```

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

The thread goes into a sleeping state, if it is not awakened explicitly within the specified system time then it is forcibly awakened with a **MSG_TIMEOUT** low level message.
### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>newstate</code></td>
<td>the new thread state or a semaphore pointer</td>
</tr>
<tr>
<td><code>timeout</code></td>
<td>the number of ticks before the operation timeouts. the following special values are allowed:</td>
</tr>
<tr>
<td></td>
<td>• <code>TIME_INFINITE</code> no timeout.</td>
</tr>
</tbody>
</table>

### Returns

The wakeup message.

### Return values

- `MSG_TIMEOUT` 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:

```
chSchGoSleepTimeoutS
  `-- chDbgCheckClassS
  `-- chTimeIsInRangeX
     `-- chSysHalt
```

### 6.3.4.28 chTimeIsInRangeX()

```c
bool chTimeIsInRangeX (  
    systime_t time,  
    systime_t start,  
    systime_t end )
```

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

### Note

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

<table>
<thead>
<tr>
<th>in</th>
<th>time</th>
<th>the time to be verified</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>start</td>
<td>the start of the time window (inclusive)</td>
</tr>
<tr>
<td>in</td>
<td>end</td>
<td>the end of the time window (non inclusive)</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>true</th>
<th>current time within the specified time window.</th>
</tr>
</thead>
<tbody>
<tr>
<td>false</td>
<td>current time not within the specified time window.</td>
</tr>
</tbody>
</table>

Function Class:

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

### 6.3.4.29 chThdCreateI()

```c
thread_t * chThdCreateI ( 
    const thread_descriptor_t * tdp )
```

Creates a new thread into a static memory area.

The new thread is initialized and make ready to execute.

Note

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

Parameters

| out | tdp | pointer to the thread descriptor structure |

Returns

The pointer to the `thread_t` structure allocated for the thread.
Function Class:

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

Here is the call graph for this function:

6.3.4.30 chThdCreate()

```c
thread_t * chThdCreate (
    const thread_descriptor_t * tdp )
```

Creates a new thread into a static memory area.

The new thread is initialized and made ready to execute.

Note

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

Parameters

- **out tdp** pointer to the thread descriptor structure

Returns

The pointer to the `thread_t` structure allocated for the thread.
Function Class:

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

Here is the call graph for this function:

6.3.4.31 chThdExit()

```c
void chThdExit (  
    msg_t msg )
```

Terminates the current thread.

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

Postcondition

Exiting a non-static thread that does not have references (detached) causes the thread to remain in the registry. It can only be removed by performing a registry scan operation.

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

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>msg</th>
<th>thread exit code</th>
</tr>
</thead>
</table>
6.3.4.32 chThdWait()

```c
msg_t chThdWait ( thread_t * tp )
```

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

**Parameters**
- **in tp** pointer to the thread

**Returns**
- The exit code from the terminated thread.

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

6.3.4.33 chThdSuspendTimeoutS()

```c
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

<table>
<thead>
<tr>
<th>in</th>
<th>trp</th>
<th>a pointer to a thread reference object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>timeout</td>
<td>the number of ticks before the operation timeouts, the following special values are allowed:</td>
</tr>
<tr>
<td></td>
<td>TIME_IMMEDIATE immediate timeout.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>TIME_INFINITE no timeout.</td>
<td></td>
</tr>
</tbody>
</table>

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:

6.3.4.34 chThdResumeI()

```c
void chThdResumeI ( 
    thread_reference_t * trp, 
    msg_t msg )
```

Wakes up a thread waiting on a thread reference object.

Note

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

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>trp</th>
<th>a pointer to a thread reference object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>msg</td>
<td>the message code</td>
</tr>
</tbody>
</table>
Function Class:

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

### 6.3.4.35 chThdResume()

```c
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   | trp | a pointer to a thread reference object |
| in   | msg | the message code |

Function Class:

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

### 6.3.4.36 chThdSleep()

```c
void chThdSleep (  
    sysinterval_t timeout )
```

Suspends the invoking thread for the specified time.

**Parameters**

| in       | timeout | 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.
6.3.4.37 chThdSleepUntil()

```c
void chThdSleepUntil (
   systime_t abstime )
```

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

**Parameters**

| in | abstime | absolute system time |

**Function Class:**

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

6.3.4.38 chThdEnqueueTimeoutS()

```c
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 | tqp | pointer to the threads queue object |
| in | timeout | the timeout in system ticks, the special values are handled as follow: |
|    |         | • TIME_IMMEDIATE immediate timeout. |
|    |         | • TIME_INFINITE no timeout. |

**Returns**

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

**Return values**

| MSG_TIMEOUT | if the thread has not been dequeued within the specified timeout or if the function has been invoked with TIME_IMMEDIATE 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:

![Call Graph](call_graph.png)

### 6.3.4.39 chThdDoDequeueNextI()

```c
void chThdDoDequeueNextI(
    threads_queue_t * tqp,
    msg_t msg)
```

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

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

**Precondition**

The queue must contain at least an object.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>tqp</th>
<th>pointer to the threads queue object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>msg</td>
<td>the message code</td>
</tr>
</tbody>
</table>

**Function 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:

![Call Graph](call_graph2.png)
### 6.3.4.40  chThdDequeueNextI()

```c
void chThdDequeueNextI (
    threads_queue_t * tqp,
    msg_t msg )
```

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

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>tqp</code></td>
<td>pointer to the threads queue object</td>
</tr>
<tr>
<td><code>msg</code></td>
<td>the message code</td>
</tr>
</tbody>
</table>

**Function 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:

![Call graph for chThdDequeueNextI()](image)

### 6.3.4.41  chThdDequeueAllI()

```c
void chThdDequeueAllI (
    threads_queue_t * tqp,
    msg_t msg )
```

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

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>tqp</code></td>
<td>pointer to the threads queue object</td>
</tr>
<tr>
<td><code>msg</code></td>
<td>the message code</td>
</tr>
</tbody>
</table>
Function 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:

![Call Graph](image)

6.3.5 Variable Documentation

6.3.5.1 nil

```
os_instance_t nil
```

System data structures.
6.4 Semaphores

6.4.1 Detailed Description

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 chSemObjectInit(sp, n) ((sp)->cnt = (n))
  Initializes a semaphore with the specified counter value.
• #define chSemReset(sp, n) chSemResetWithMessage(sp, n, MSG_RESET)
  Performs a reset operation on the semaphore.
• #define chSemResetI(sp, n) chSemResetWithMessageI(sp, n, MSG_RESET)
  Performs a reset operation on the semaphore.
• #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.

Functions

• 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 chSemResetWithMessage (semaphore_t *sp, cnt_t n, msg_t msg)
  Performs a reset operation on the semaphore.
• void chSemResetWithMessageI (semaphore_t *sp, cnt_t n, msg_t msg)
  Performs a reset operation on the semaphore.

6.4.2 Macro Definition Documentation
6.4.2.1 __SEMAPHORE_DATA

#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

<table>
<thead>
<tr>
<th>in</th>
<th>name</th>
<th>the name of the semaphore variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>n</td>
<td>the counter initial value, this value must be non-negative</td>
</tr>
</tbody>
</table>

6.4.2.2 SEMAPHORE_DECL

#define SEMAPHORE_DECL(
    name,
    n ) semaphore_t name = __SEMAPHORE_DATA(name, n)

Static semaphore initializer.

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

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>name</th>
<th>the name of the semaphore variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>n</td>
<td>the counter initial value, this value must be non-negative</td>
</tr>
</tbody>
</table>

6.4.2.3 chSemObjectInit

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

Initializes a semaphore with the specified counter value.

Parameters

<table>
<thead>
<tr>
<th>out</th>
<th>sp</th>
<th>pointer to a semaphore_t structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>n</td>
<td>initial value of the semaphore counter. Must be non-negative.</td>
</tr>
</tbody>
</table>
6.4 Semaphores

Function Class:

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

6.4.2.4 chSemReset

#define chSemReset(sp, n) chSemResetWithMessage(sp, n, MSG_RESET)

Performs a reset operation on the semaphore.

Postcondition

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

Note

This function implicitly sends MSG_RESET as message.

Parameters

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

Function Class:

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

6.4.2.5 chSemResetI

#define chSemResetI(sp, n) chSemResetWithMessageI(sp, n, MSG_RESET)

Performs a reset operation on the semaphore.

Postcondition

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

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

Note

This function implicitly sends MSG_RESET as message.
Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>sp</th>
<th>pointer to a semaphore_t structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>n</td>
<td>the new value of the semaphore counter. The value must be non-negative.</td>
</tr>
</tbody>
</table>

Function Class:

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

6.4.2.6 chSemWait

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

Performs a wait operation on a semaphore.

Parameters

| in | sp | pointer to a semaphore_t structure |

Returns

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

Return values

| CH_MSG_OK | if the thread has not stopped on the semaphore or the semaphore has been signaled. |
| CH_MSG_RST | if the semaphore has been reset using chSemReset(). |

Function Class:

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

6.4.2.7 chSemWaitS

```c
#define chSemWaitS( 
    sp ) chSemWaitTimeoutS(sp, TIME_INFINITE)
```

Performs a wait operation on a semaphore.

Parameters

| in | sp | pointer to a semaphore_t structure |
Returns

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

Return values

<table>
<thead>
<tr>
<th>CH_MSG_OK</th>
<th>if the thread has not stopped on the semaphore or the semaphore has been signaled.</th>
</tr>
</thead>
<tbody>
<tr>
<td>CH_MSG_RST</td>
<td>if the semaphore has been reset using chSemReset().</td>
</tr>
</tbody>
</table>

Function Class:

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

6.4.2.8 chSemFastWaitI

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

Decreases the semaphore counter.

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

Parameters

| in | sp | pointer to a 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.

6.4.2.9 chSemFastSignalI

#define chSemFastSignalI(
    sp ) ((sp)->cnt++)

Increases the semaphore counter.

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

Parameters

| in | sp | pointer to a 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.

6.4.2.10 chSemGetCounterI

```
#define chSemGetCounterI(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.

6.4.3 Function Documentation

6.4.3.1 chSemWaitTimeout()

```
msg_t chSemWaitTimeout ( 
    semaphore_t * sp,
    sysinterval_t timeout 
)
```

Performs a wait operation on a semaphore with timeout specification.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>sp</th>
<th>pointer to a semaphore_t structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>timeout</td>
<td>the number of ticks before the operation times out, the following special values are allowed:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• TIME_IMMEDIATE immediate timeout.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• TIME_INFINITE no timeout.</td>
</tr>
</tbody>
</table>

Returns

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

Return values

<table>
<thead>
<tr>
<th>NIL_MSG_OK</th>
<th>if the thread has not stopped on the semaphore or the semaphore has been signaled.</th>
</tr>
</thead>
<tbody>
<tr>
<td>NIL_MSG_RST</td>
<td>if the semaphore has been reset using chSemReset ().</td>
</tr>
<tr>
<td>NIL_MSG_TMO</td>
<td>if the semaphore has not been signaled or reset within the specified timeout.</td>
</tr>
</tbody>
</table>
6.4 Semaphores

Function Class:

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

Here is the call graph for this function:

![Call Graph](image)

6.4.3.2 chSemWaitTimeoutS()

```c
msg_t chSemWaitTimeoutS (  
    semaphore_t * sp,  
    sysinterval_t timeout )
```

Performs a wait operation on a semaphore with timeout specification.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>sp</th>
<th>pointer to a semaphore_t structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>timeout</td>
<td>the number of ticks before the operation timeouts, the following special values are allowed:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• TIME_IMMEDIATE immediate timeout.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• TIME_INFINITE no timeout.</td>
</tr>
</tbody>
</table>

Returns

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

Return values

- **NIL_MSG_OK** if the thread has not stopped on the semaphore or the semaphore has been signaled.
- **NIL_MSG_RST** if the semaphore has been reset using `chSemReset()`.
- **NIL_MSG_TMO** if the semaphore has not been signaled or reset within the specified timeout.
Function Class:

This is a **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:

```
chSemWaitTimeoutS
  \-- chDbgCheckClassS
     \-- chSysHalt

chSchGoSleepTimeoutS
  \-- chTimeInRangeX
```

### 6.4.3.3 chSemSignal()

```c
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:

```
chSemSignal
  \-- chSemSignal
     \-- chSemSignal
         \-- chSemSignal
             \-- chDbgCheckClassS
                 \-- chDbgCheckClassS
```

ChibiOS/NIL
6.4 Semaphores

6.4.3.4 chSemSignalI()

```c
void chSemSignalI ( 
    semaphore_t * sp )
```

Performs a signal operation on a semaphore.

**Postcondition**

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

**Parameters**

| in  | 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:

![Call Graph](diagram.png)

6.4.3.5 chSemResetWithMessage()

```c
void chSemResetWithMessage ( 
    semaphore_t * sp, 
    cnt_t n, 
    msg_t msg )
```

Performs a reset operation on the semaphore.

**Postcondition**

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

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

<table>
<thead>
<tr>
<th>in</th>
<th>sp</th>
<th>pointer to a semaphore_t structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>n</td>
<td>the new value of the semaphore counter. The value must be non-negative.</td>
</tr>
<tr>
<td>in</td>
<td>msg</td>
<td>message to be sent</td>
</tr>
</tbody>
</table>

Function Class:

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

Here is the call graph for this function:

6.4.3.6 chSemResetWithMessage()  

```c
void chSemResetWithMessageI (  
    semaphore_t * sp,  
    cnt_t n,  
    msg_t msg )
```

Performs a reset operation on the semaphore.

Postcondition

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

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

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>sp</th>
<th>pointer to a semaphore_t structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>n</td>
<td>the new value of the semaphore counter. The value must be non-negative.</td>
</tr>
<tr>
<td>in</td>
<td>msg</td>
<td>message to be sent</td>
</tr>
</tbody>
</table>
Function 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:
6.5 Events

6.5.1 Detailed Description

Macros

- **#define ALL_EVENTS ((eventmask_t)-1)**
  
  All events allowed mask.

- **#define EVENT_MASK(eid) ((eventmask_t)1 << (eventmask_t)(eid))**
  
  Returns an event mask from an event identifier.

- **#define _EVENTSOURCE_DATA(name) {(event_listener_t ∗)(name)}**
  
  Data part of a static event source initializer.

- **#define EVENTSOURCE_DECL(name) event_source_t name = _EVENTSOURCE_DATA(name)**
  
  Static event source initializer.

Macro Functions

- **#define chEvtObjectInit(esp)**
  
  Initializes an Event Source.

- **#define chEvtRegisterMask(esp, elp, events) chEvtRegisterMaskWithFlags(esp, elp, events, (eventflags_t)-1)**
  
  Registers an Event Listener on an Event Source.

- **#define chEvtRegister(esp, elp, event) chEvtRegisterMask(esp, elp, EVENT_MASK(event))**
  
  Registers an Event Listener on an Event Source.

- **#define chEvtIsListeningI(esp)**
  
  Verifies if there is at least one event_listener_t registered.

- **#define chEvtBroadcast(esp) chEvtBroadcastFlags(esp, (eventflags_t)0)**
  
  Signals all the Event Listeners registered on the specified Event Source.

- **#define chEvtBroadcastI(esp) chEvtBroadcastFlagsI(esp, (eventflags_t)0)**
  
  Signals all the Event Listeners registered on the specified Event Source.

- **#define chEvtAddEventsI(events)**
  
  Adds (OR) a set of events to the current thread, this is much faster than using chEvtBroadcast() or chEvtSignal().

- **#define chEvtGetEventsX(void)**
  
  Returns the events mask.

- **#define chEvtWaitOne(events) chEvtWaitOneTimeout(events, TIME_INFINITE)**
  
  Waits for exactly one of the specified events.

- **#define chEvtWaitAny(events) chEvtWaitAnyTimeout(events, TIME_INFINITE)**
  
  Waits for any of the specified events.

- **#define chEvtWaitAll(events) chEvtWaitAllTimeout(events, TIME_INFINITE)**
  
  Waits for all the specified events.

Typedefs

- **typedef struct event_source event_source_t**
  
  Event Source structure.

- **typedef void(* evhandler_t)(eventid_t id)**
  
  Event Handler callback function.
6.5 Events

Data Structures

- struct event_listener
  
  Event Listener structure.

- struct event_source
  
  Event Source structure.

Functions

- void chEvtRegisterMaskWithFlags (event_source_t *esp, event_listener_t *elp, eventmask_t events, eventflags_t wflags)
  
  Registers an Event Listener on an Event Source.

- void chEvtUnregister (event_source_t *esp, event_listener_t *elp)
  
  Unregisters an Event Listener from its Event Source.

- eventmask_t chEvtGetAndClearEventsI (eventmask_t events)
  
  Clears the pending events specified in the events mask.

- eventmask_t chEvtGetAndClearEvents (eventmask_t events)
  
  Clears the pending events specified in the events mask.

- eventmask_t chEvtAddEvents (eventmask_t events)
  
  Adds (OR) a set of events to the current thread, this is much faster than using chEvtBroadcast() or chEvtSignal().

- void chEvtBroadcastFlagsI (event_source_t *esp, eventflags_t flags)
  
  Signals all the Event Listeners registered on the specified Event Source.

- eventflags_t chEvtGetAndClearFlags (event_listener_t *elp)
  
  Returns the flags associated to an event_listener_t.

- void chEvtSignal (thread_t *tp, eventmask_t events)
  
  Adds a set of event flags directly to the specified thread_t.

- void chEvtSignalAll (thread_t *tp, eventmask_t events)
  
  Adds a set of event flags directly to the specified thread_t.

- void chEvtBroadcastFlags (event_source_t *esp, eventflags_t flags)
  
  Signals all the Event Listeners registered on the specified Event Source.

- eventflags_t chEvtGetAndClearFlagsI (event_listener_t *elp)
  
  Returns the unmasked flags associated to an event_listener_t.

- void chEvtDispatch (const evhandler_t *handlers, eventmask_t events)
  
  Invokes the event handlers associated to an event flags mask.

- eventmask_t chEvtWaitOneTimeout (eventmask_t events, sysinterval_t timeout)
  
  Waits for exactly one of the specified events.

- eventmask_t chEvtWaitAnyTimeout (eventmask_t mask, sysinterval_t timeout)
  
  Waits for any of the specified events.

- eventmask_t chEvtWaitAllTimeout (eventmask_t mask, sysinterval_t timeout)
  
  Waits for all the specified events.

6.5.2 Macro Definition Documentation
6.5.2.1  ALL_EVENTS

#define ALL_EVENTS ((eventmask_t)-1)

All events allowed mask.

6.5.2.2  EVENT_MASK

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

Returns an event mask from an event identifier.

6.5.2.3  _EVENTSOURCE_DATA

#define _EVENTSOURCE_DATA(name) {(event_listener_t ∗)(name)}

Data part of a static event source initializer.

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

Parameters

| name | the name of the event source variable |

6.5.2.4  EVENTSOURCE_DECL

#define EVENTSOURCE_DECL(name) event_source_t name = _EVENTSOURCE_DATA(name)

Static event source initializer.

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

Parameters

| name | the name of the event source variable |
6.5.2.5  chEvtObjectInit

#define chEvtObjectInit( 
    esp )

Value:

    do { 
       (esp)->next = (event_listener_t *)(esp); 
    while (0)

Initializes an Event Source.

Note

This function can be invoked before the kernel is initialized because it just prepares an event_source_t structure.

Parameters

| in  | esp | pointer to the event_source_t structure |

Function Class:

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

6.5.2.6  chEvtRegisterMask

#define chEvtRegisterMask( 
    esp, 
    elp, 
    events ) chEvtRegisterMaskWithFlags(esp, elp, events, (eventflags_t)-1)

Registers an Event Listener on an Event Source.

Once a thread has registered as listener on an event source it will be notified of all events broadcasted there.

Note

Multiple Event Listeners can specify the same bits to be ORed to different threads.

Parameters

| in   | esp | pointer to the event_source_t structure |
| out  | elp | pointer to the event_listener_t structure |
| in   | events | the mask of events to be ORed to the thread when the event source is broadcasted |

Function Class:

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

```c
#define chEvtRegister(
    esp,
    elp,
    event ) chEvtRegisterMask(esp, elp, EVENT_MASK(event))
```

Registers an Event Listener on an Event Source.

**Note**

Multiple Event Listeners can use the same event identifier, the listener will share the callback function.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>esp</th>
<th>pointer to the event_source_t structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>elp</td>
<td>pointer to the event_listener_t structure</td>
</tr>
<tr>
<td>in</td>
<td>event</td>
<td>numeric identifier assigned to the Event Listener. The value must range between zero and the size, in bit, of the eventmask_t type minus one.</td>
</tr>
</tbody>
</table>

**Function Class:**

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

### 6.5.2.8 chEvtIsListeningI

```c
#define chEvtIsListeningI(
    esp ) (bool)((esp) != (event_source_t *(esp)->next)
```

Verifies if there is at least one event_listener_t registered.

**Parameters**

| in   | esp   | pointer to the event_source_t structure |

**Returns**

The event source status.

**Function Class:**

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

#define chEvtBroadcast( 
    esp ) chEvtBroadcastFlags(esp, (eventflags_t)0)

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

Parameters

| in  | esp | pointer to the event_source_t structure |

Function Class:

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

6.5.2.10 chEvtBroadcastI

#define chEvtBroadcastI( 
    esp ) chEvtBroadcastFlagsI(esp, (eventflags_t)0)

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

Postcondition

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

Parameters

| in  | esp | pointer to the event_source_t structure |

Function Class:

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

6.5.2.11 chEvtAddEventsI

#define chEvtAddEventsI( 
    events ) (nil.current->epmask |= events)

Adds (OR) a set of events to the current thread, this is much faster than using chEvtBroadcast() or chEvtSignal().
Parameters

| in   | events | the events to be added |

Returns

The mask of currently pending events.

Function Class:

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

6.5.2.12 chEvtGetEventsX

#define chEvtGetEventsX(
    void ) (nil.current->epmask)

Returns the events mask.

The pending events mask is returned but not altered in any way.

Returns

The pending events mask.

Function Class:

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

6.5.2.13 chEvtWaitOne

#define chEvtWaitOne(
    events ) chEvtWaitOneTimeout(events, TIME_INFINITE)

Waits for exactly one of the specified events.

The function waits for one event among those specified in events to become pending then the event is cleared and returned.

Note

One and only one event is served in the function, the one with the lowest event id. The function is meant to be invoked into a loop in order to serve all the pending events.

This means that Event Listeners with a lower event identifier have an higher priority.
6.5 Events

Parameters

| in | events | events that the function should wait for, ALL_EVENTS enables all the events |

Returns

The mask of the lowest event id served and cleared.

Return values

| 0 | if the operation has timed out. |

Function Class:

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

6.5.2.14 chEvtWaitAny

#define chEvtWaitAny(
    events ) chEvtWaitAnyTimeout(events, TIME_INFINITE)

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 | events | events that the function should wait for, ALL_EVENTS enables all the events |

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.
6.5.2.15 chEvtWaitAll

#define chEvtWaitAll(
    events ) chEvtWaitAllTimeout(events, TIME_INFINITE)

Waits for all the specified events.

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

Parameters

| in | events | events that the function should wait for. ALL_EVENTS enables all the events |

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.

6.5.3 Typedef Documentation

6.5.3.1 event_source_t

typedef struct event_source event_source_t

Event Source structure.

6.5.3.2 evhandler_t

typedef void(* evhandler_t) (eventid_t id)

Event Handler callback function.

6.5.4 Function Documentation
6.5 Events

6.5.4.1 chEvtRegisterMaskWithFlags()

```c
void chEvtRegisterMaskWithFlags (  
    event_source_t * esp,  
    event_listener_t * elp,  
    eventmask_t events,  
    eventflags_t wflags )
```

Registers an Event Listener on an Event Source.

Once a thread has registered as listener on an event source it will be notified of all events broadcasted there.

**Note**

Multiple Event Listeners can specify the same bits to be ORed to different threads.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>esp</th>
<th>pointer to the event_source_t structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>elp</td>
<td>pointer to the event_listener_t structure</td>
</tr>
<tr>
<td>in</td>
<td>events</td>
<td>events to be ORed to the thread when the event source is broadcasted</td>
</tr>
<tr>
<td>in</td>
<td>wflags</td>
<td>mask of flags the listening thread is interested in</td>
</tr>
</tbody>
</table>

**Function Class:**

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

6.5.4.2 chEvtUnregister()

```c
void chEvtUnregister (  
    event_source_t * esp,  
    event_listener_t * elp )
```

Unregisters an Event Listener from its Event Source.

**Note**

If the event listener is not registered on the specified event source then the function does nothing.

For optimal performance it is better to perform the unregister operations in inverse order of the register operations (elements are found on top of the list).

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>esp</th>
<th>pointer to the event_source_t structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>elp</td>
<td>pointer to the event_listener_t structure</td>
</tr>
</tbody>
</table>
Function Class:

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

### 6.5.4.3 chEvtGetAndClearEventsI()

```c
eventmask_t chEvtGetAndClearEventsI (  
    eventmask_t events )
```

Clears the pending events specified in the events mask.

**Parameters**

| in  | events | the events to be cleared |

**Returns**

The mask of pending events that were cleared.

Function Class:

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

### 6.5.4.4 chEvtGetAndClearEvents()

```c
eventmask_t chEvtGetAndClearEvents (  
    eventmask_t events )
```

Clears the pending events specified in the events mask.

**Parameters**

| in  | events | the events to be cleared |

**Returns**

The mask of pending events that were cleared.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.
Here is the call graph for this function:

![Call Graph](call_graph.png)

### 6.5.4.5 chEvtAddEvents()

```c
eventmask_t chEvtAddEvents (  
    eventmask_t events  
)
```

Adds (OR) a set of events to the current thread, this is much faster than using `chEvtBroadcast()` or `chEvtSignal()`.

**Parameters**

- **in events** the events to be added

**Returns**

The mask of currently pending events.

**Function Class:**

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

### 6.5.4.6 chEvtBroadcastFlagsI()

```c
void chEvtBroadcastFlagsI (  
    event_source_t * esp,  
    eventflags_t flags  
)
```

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

This function variants ORs the specified event flags to all the threads registered on the `event_source_t` in addition to the event flags specified by the threads themselves in the `event_listener_t` objects.

**Postcondition**

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

| in | esp | pointer to the event_source_t structure |
| in | flags | the flags set to be added to the listener flags mask |

Function Class:

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

Here is the call graph for this function:

![Call Graph](image)

6.5.4.7 chEvtGetAndClearFlags()

```c
eventflags_t chEvtGetAndClearFlags (event_listener_t *elp)
```

Returns the flags associated to an event_listener_t.

The flags are returned and the event_listener_t flags mask is cleared.

Parameters

| in | elp | pointer to the event_listener_t structure |

Returns

The flags added to the listener by the associated event source.

Function Class:

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

6.5.4.8 chEvtSignal()

```c
void chEvtSignal ( 
    thread_t * tp, 
    eventmask_t events )
```

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

- **in tp** the thread to be signaled
- **in events** 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:

![Call Graph](attachment:image.png)

### 6.5.4.9 chEvtSignalI()

```c
void chEvtSignalI (
    thread_t * tp,
    eventmask_t events )
```

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

Postcondition

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

Parameters

- **in tp** the thread to be signaled
- **in events** 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.
6.5 Events

Here is the call graph for this function:

```
    chEvtSignalI  chDbgCheckClassI  chSysHalt
```

6.5.4.10 chEvtBroadcastFlags()

```c
void chEvtBroadcastFlags (
    event_source_t ∗ esp,
    eventflags_t flags)
```

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

This function ORs the specified event flags to all the threads registered on the event_source_t in addition to the event flags specified by the threads themselves in the event_listener_t objects.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>esp</th>
<th>pointer to the event_source_t structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>flags</td>
<td>the flags set to be added to the listener flags mask</td>
</tr>
</tbody>
</table>

Function Class:

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

Here is the call graph for this function:

```
    chEvtBroadcastFlags  chEvtSignal  chDbgCheckClass

    chEvtBroadcastFlags  chSchRescheduleS  chDbgCheckClassS

    chDbgCheckClassS  chSchDoPreemption
```
6.5.4.11 chEvtGetAndClearFlagsI()

```
eventflags_t chEvtGetAndClearFlagsI ( 
    event_listener_t * elp )
```

Returns the unmasked flags associated to an `event_listener_t`.

The flags are returned and the `event_listener_t` flags mask is cleared.

**Parameters**

| in  | elp | pointer to the event_listener_t structure |

**Returns**

The flags added to the listener by the associated event source.

**Function Class:**

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

6.5.4.12 chEvtDispatch()

```
void chEvtDispatch ( 
    const evhandler_t * handlers, 
    eventmask_t events )
```

Invokes the event handlers associated to an event flags mask.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>events</th>
<th>mask of events to be dispatched</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>handlers</td>
<td>an array of evhandler_t. The array must have size equal to the number of bits in eventmask_t.</td>
</tr>
</tbody>
</table>

**Function Class:**

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

6.5.4.13 chEvtWaitOneTimeout()

```
eventmask_t chEvtWaitOneTimeout ( 
    eventmask_t events, 
    sysinterval_t timeout )
```
Waits for exactly one of the specified events.

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

**Note**

One and only one event is served in the function, the one with the lowest event id. The function is meant to be invoked into a loop in order to serve all the pending events.
This means that Event Listeners with a lower event identifier have an higher priority.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th><code>events</code></th>
<th>events that the function should wait for, <code>ALL_EVENTS</code> enables all the events</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td><code>timeout</code></td>
<td>the number of ticks before the operation timeouts, the following special values are allowed:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <code>TIME_IMMEDIATE</code> immediate timeout.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <code>TIME_INFINITE</code> no timeout.</td>
</tr>
</tbody>
</table>

**Returns**

The mask of the lowest event id served and cleared.

**Return values**

- `0` if the operation has timed out.

**Function Class:**

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

Here is the call graph for this function:

![Call Graph](image)

### 6.5.4.14 chEvtWaitAnyTimeout()

```
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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong><code>mask</code></strong></td>
<td>mask of the event flags that the function should wait for, <code>ALL_EVENTS</code> enables all the events</td>
</tr>
<tr>
<td><strong><code>timeout</code></strong></td>
<td>the number of ticks before the operation timeouts, the following special values are allowed:</td>
</tr>
<tr>
<td></td>
<td>• <code>TIME_IMMEDIATE</code> immediate timeout.</td>
</tr>
<tr>
<td></td>
<td>• <code>TIME_INFINITE</code> no timeout.</td>
</tr>
</tbody>
</table>

**Returns**

The mask of the served and cleared events.

**Return values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>if the operation has timed out.</td>
</tr>
</tbody>
</table>

**Function Class:**

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

Here is the call graph for this function:

![Call Graph](image)

### 6.5.4.15 chEvtWaitAllTimeout()

```c
eventmask_t chEvtWaitAllTimeout ( 
    eventmask_t mask, 
    sysinterval_t timeout )
```

Waits for all the specified events.

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

Parameters

| in | mask | mask of the event flags that the function should wait for, **ALL_EVENTS** enables all the events |
| in | timeout | the number of ticks before the operation timeouts, the following special values are allowed: |
|    |        | • **TIME_IMMEDIATE** immediate timeout. |
|    |        | • **TIME_INFINITE** no timeout. |

Returns

The 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:
6.6 Synchronous Messages

6.6.1 Detailed Description

Macro Functions

• \#define chMsgWaitS() chMsgWaitTimeoutS(TIME_INFINITE)
  Suspends the thread and waits for an incoming message.
• \#define chMsgGet(tp) ((tp)->sntmsg)
  Returns the message carried by the specified thread.
• \#define chMsgReleaseS(tp, msg)
  Releases the thread waiting on top of the messages queue.

Functions

• msg_t chMsgSend (thread_t *tp, msg_t msg)
  Sends a message to the specified thread.
• thread_t * chMsgWait (void)
  Suspends the thread and waits for an incoming message.
• thread_t * chMsgWaitTimeout (sysinterval_t timeout)
  Suspends the thread and waits for an incoming message or a timeout to occur.
• thread_t * chMsgWaitTimeoutS (sysinterval_t timeout)
  Suspends the thread and waits for an incoming message or a timeout to occur.
• void chMsgRelease (thread_t *tp, msg_t msg)
  Releases a sender thread specifying a response message.

6.6.2 Macro Definition Documentation

6.6.2.1 chMsgWaitS

\#define chMsgWaitS() chMsgWaitTimeoutS(TIME_INFINITE)
Suspends the thread and waits for an incoming message.

Postcondition

After receiving a message the function chMsgGet() must be called in order to retrieve the message and then chMsgRelease() must be invoked in order to acknowledge the reception and send the answer.

Note

If the message is a pointer then you can assume that the data pointed by the message is stable until you invoke chMsgRelease() because the sending thread is suspended until then.

The reference counter of the sender thread is not increased, the returned pointer is a temporary reference.

Returns

A pointer to the thread carrying the message.

Function Class:

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

6.6.2.2 chMsgGet

```c
#define chMsgGet(tp) (((tp)->sntmsg)
```

Returns the message carried by the specified thread.

Precondition

This function must be invoked immediately after exiting a call to `chMsgWait()`.

Parameters

- `tp` pointer to the thread

Returns

The message carried by the sender.

Function Class:

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

6.6.2.3 chMsgReleaseS

```c
#define chMsgReleaseS(tp, msg)
```

Value:

```c
do {
  (void) chSchReadyI(tp, msg);
  chSchRescheduleS();
} while (false)
```

Releases the thread waiting on top of the messages queue.

Precondition

Invoke this function only after a message has been received using `chMsgWait()`.

Parameters

- `tp` pointer to the thread
- `msg` message to be returned to the sender
Function Class:

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

### 6.6.3 Function Documentation

#### 6.6.3.1 chMsgSend()

```c
msg_t chMsgSend (        
    thread_t * tp,       
    msg_t msg )
```

Sends a message to the specified thread.

The sender is stopped until the receiver executes a `chMsgRelease()` after receiving the message.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>tp</th>
<th>the pointer to the thread</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>msg</td>
<td>the message</td>
</tr>
</tbody>
</table>

**Returns**

The answer message from `chMsgRelease()`.

Function Class:

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

#### 6.6.3.2 chMsgWait()

```c
thread_t * chMsgWait (    
    void )
```

Suspends the thread and waits for an incoming message.

**Postcondition**

After receiving a message the function `chMsgGet()` must be called in order to retrieve the message and then `chMsgRelease()` must be invoked in order to acknowledge the reception and send the answer.

**Note**

If the message is a pointer then you can assume that the data pointed by the message is stable until you invoke `chMsgRelease()` because the sending thread is suspended until then.

The reference counter of the sender thread is not increased, the returned pointer is a temporary reference.
6.6 Synchronous Messages

Returns

A pointer to the thread carrying the message.

Function Class:

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

6.6.3.3 chMsgWaitTimeout()

\[
\text{thread}_t \ast \text{chMsgWaitTimeout} ( \\
\quad \text{sysinterval}_t \ \text{timeout} )
\]

Suspends the thread and waits for an incoming message or a timeout to occur.

Postcondition

After receiving a message the function \text{chMsgGet()} must be called in order to retrieve the message and then \text{chMsgRelease()} must be invoked in order to acknowledge the reception and send the answer.

Note

If the message is a pointer then you can assume that the data pointed by the message is stable until you invoke \text{chMsgRelease()} because the sending thread is suspended until then.

The reference counter of the sender thread is not increased, the returned pointer is a temporary reference.

Parameters

<table>
<thead>
<tr>
<th>\text{in}</th>
<th>\text{timeout}</th>
<th>the number of ticks before the operation timeouts, the following special values are allowed:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>• \text{TIME_IMMEDIATE} immediate timeout.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• \text{TIME_INFINITE} no timeout.</td>
</tr>
</tbody>
</table>

Returns

A pointer to the thread carrying the message.

Return values

| \text{NULL} | if a timeout occurred. |
Function Class:

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

Here is the call graph for this function:

![Call Graph](image)

6.6.3.4  chMsgWaitTimeoutS()

```c
thread_t * chMsgWaitTimeoutS ( 
    sysinterval_t timeout )
```

Suspends the thread and waits for an incoming message or a timeout to occur.

Postcondition

After receiving a message the function `chMsgGet()` must be called in order to retrieve the message and then `chMsgRelease()` must be invoked in order to acknowledge the reception and send the answer.

Note

If the message is a pointer then you can assume that the data pointed by the message is stable until you invoke `chMsgRelease()` because the sending thread is suspended until then.

The reference counter of the sender thread is not increased, the returned pointer is a temporary reference.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>timeout</th>
<th>the number of ticks before the operation timeouts, the following special values are allowed:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>• <code>TIME_INFINITE</code> no timeout.</td>
</tr>
</tbody>
</table>

Returns

A pointer to the thread carrying the message.

Return values

- `NULL` 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:

![Call Graph](image)

### 6.6.3.5 chMsgRelease()

```c
void chMsgRelease (
    thread_t * tp,
    msg_t msg )
```

Releases a sender thread specifying a response message.

**Precondition**

Invoke this function only after a message has been received using `chMsgWait()`.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>tp</th>
<th>pointer to the thread</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>msg</td>
<td>message to be returned to the sender</td>
</tr>
</tbody>
</table>

**Function Class:**

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

6.7.1 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
6.8 Version Numbers and Identification

6.8.1 Detailed Description

OS Library related info.

Macros

- `#define __CHIBIOS_OSLIB__`
  ChibiOS/LIB identification macro.
- `#define CH_OSLIB_STABLE 0`
  Stable release flag.

ChibiOS/LIB version identification

- `#define CH_OSLIB_VERSION "1.3.0"
  OS Library version string.`
- `#define CH_OSLIB_MAJOR 1`
  OS Library version major number.
- `#define CH_OSLIB_MINOR 3`
  OS Library version minor number.
- `#define CH_OSLIB_PATCH 0`
  OS Library version patch number.

Functions

- `static void __oslib_init (void)`
  *Initialization of all library modules.*

6.8.2 Macro Definition Documentation

6.8.2.1 __CHIBIOS_OSLIB__

`#define __CHIBIOS_OSLIB__`
ChibiOS/LIB identification macro.

6.8.2.2 CH_OSLIB_STABLE

`#define CH_OSLIB_STABLE 0`
Stable release flag.
6.8.2.3  CH_OSLIB_VERSION

#define CH_OSLIB_VERSION "1.3.0"

OS Library version string.

6.8.2.4  CH_OSLIB_MAJOR

#define CH_OSLIB_MAJOR 1

OS Library version major number.

6.8.2.5  CH_OSLIB_MINOR

#define CH_OSLIB_MINOR 3

OS Library version minor number.

6.8.2.6  CH_OSLIB_PATCH

#define CH_OSLIB_PATCH 0

OS Library version patch number.

6.8.3  Function Documentation
6.8.3.1 __oslib_init()

static void __oslib_init (  
    void ) [inline], [static]

Initialization of all library modules.

Function Class:

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

Here is the call graph for this function:
6.9 Synchronization

6.9.1 Detailed Description

Synchronization services.

Modules

- Binary Semaphores
- Mailboxes
- Pipes
- Delegate Threads
- Jobs Queues
6.10 Binary Semaphores

6.10.1 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 chBSemGetStateI (const binary_semaphore_t *bsp)
  
  Returns the binary semaphore current state.

6.10.2 Macro Definition Documentation
6.10.2.1 __BSEMAPHORE_DATA

#define __BSEMAPHORE_DATA(
    name,
    taken ) {__SEMAPHORE_DATA(name.sem, ((taken) ? 0 : 1))}

Data part of a static semaphore initializer.

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

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>name</th>
<th>the name of the semaphore variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>taken</td>
<td>the semaphore initial state</td>
</tr>
</tbody>
</table>

6.10.2.2 BSEMAPHORE_DECL

#define BSEMAPHORE_DECL(
    name,
    taken ) binary_semaphore_t name = __BSEMAPHORE_DATA(name, taken)

Static semaphore initializer.

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

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>name</th>
<th>the name of the semaphore variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>taken</td>
<td>the semaphore initial state</td>
</tr>
</tbody>
</table>

6.10.3 Typedef Documentation

6.10.3.1 binary_semaphore_t

typedef struct ch_binary_semaphore binary_semaphore_t

Binary semaphore type.

6.10.4 Function Documentation
6.10.4.1 chBSemObjectInit()

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

Initializes a binary semaphore.

Parameters

<table>
<thead>
<tr>
<th>out</th>
<th>bsp</th>
<th>pointer to a binary_semaphore_t structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>taken</td>
<td>initial state of the binary semaphore:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• false, the initial state is not taken.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• true, the initial state is taken.</td>
</tr>
</tbody>
</table>

Function Class:

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

6.10.4.2 chBSemWait()

static msg_t chBSemWait {
    binary_semaphore_t * bsp) [inline], [static]

Wait operation on the binary semaphore.

Parameters

| in   | bsp        | pointer to a binary_semaphore_t structure |

Returns

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

Return values

<table>
<thead>
<tr>
<th>MSG_OK</th>
<th>if the binary semaphore has been successfully taken.</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSG_RESET</td>
<td>if the binary semaphore has been reset using bsemReset().</td>
</tr>
</tbody>
</table>

Function Class:

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

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

Wait operation on the binary semaphore.

**Parameters**

| in  | bsp | pointer to a binary_semaphore_t structure |

**Returns**

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

**Return values**

| MSG_OK  | if the binary semaphore has been successfully taken. |
| MSG_RESET | if the binary semaphore has been reset using bsemReset(). |

**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:

```
chBSemWaitS -> chDbgCheckClassS -> chSysHalt
```

6.10.4.4 chBSemWaitTimeoutS()

```c
static msg_t chBSemWaitTimeoutS ( binary_semaphore_t * bsp,
                                  sysinterval_t timeout ) [inline], [static]
```

Wait operation on the binary semaphore.

**Parameters**

| in  | bsp | pointer to a binary_semaphore_t structure |
| in  | timeout | the number of ticks before the operation timeouts, the following special values are allowed: |
|     |       | • TIME_IMMEDIATE immediate timeout. |
|     |       | • TIME_INFINITE no timeout. |

ChibiOS/NIL
Returns

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

Return values

| MSG_OK   | if the binary semaphore has been successfully taken. |
| MSG_RESET| if the binary semaphore has been reset using bsemReset(). |
| MSG_TIMEOUT | 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:

6.10.4.5 chBSemWaitTimeout()

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

Wait operation on the binary semaphore.

Parameters

| in   | bsp       | pointer to a binary_semaphore_t structure |
| in   | timeout   | the number of ticks before the operation timeouts, the following special values are allowed: |
|      |           | • TIME_IMMEDIATE immediate timeout. |
|      |           | • TIME_INFINITE no timeout. |

Returns

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

Return values

| MSG_OK   | if the binary semaphore has been successfully taken. |
Return values

| MSG_RESET      | if the binary semaphore has been reset using bsemReset(). |
| MSG_TIMEOUT    | 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:

```
chBSemWaitTimeout → chSemWaitTimeout → chSemWaitTimeoutS
```

### 6.10.4.6 chBSemReset()

```c
static void chBSemResetI ( 
    binary_semaphore_t * bsp,
    bool taken ) [inline], [static]
```

Reset operation on the binary semaphore.

Note

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

This function does not reschedule.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>bsp</th>
<th>pointer to a binary_semaphore_t structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>taken</td>
<td>new state of the binary semaphore</td>
</tr>
</tbody>
</table>

- **false**, the new state is not taken.
- **true**, the new state is taken.

Function Class:

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

6.10.4.7 chBSemReset()

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

Reset operation on the binary semaphore.

Note

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

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>bsp</th>
<th>pointer to a binary_semaphore_t structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>taken</td>
<td>new state of the binary semaphore</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• false, the new state is not taken.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• true, the new state is taken.</td>
</tr>
</tbody>
</table>

Function Class:

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

6.10.4.8 chBSemSignalI()

static void chBSemSignalI (  
    binary_semaphore_t * bsp ) [inline], [static]

Performs a signal operation on a binary semaphore.

Note

This function does not reschedule.
Parameters

| in  | bsp | pointer to a binary_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:

6.10.4.9 chBSemSignal()

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

Performs a signal operation on a binary semaphore.

Parameters

| in  | bsp | pointer to a binary_semaphore_t structure |

Function Class:

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

Here is the call graph for this function:
6.10 Binary Semaphores

6.10.4.10 chBSemGetStateI()

static bool chBSemGetStateI (  
    const binary_semaphore_t * bsp ) [inline], [static]

Returns the binary semaphore current state.

Parameters

| in | bsp | pointer to a binary_semaphore_t structure |

Returns

The binary semaphore current state.

Return values

| false | if the binary semaphore is not taken. |
| true  | 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:

```
chBSemGetStateI \rightarrow chDbgCheckClassI \rightarrow chSysHalt
```
6.11 Mailboxes

6.11.1 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.

6.11.2  Macro Definition Documentation

ChibiOS/NIL
6.11.2.1  __MAILBOX_DATA

#define __MAILBOX_DATA(
    name,
    buffer,
    size )

Value:

\{
    (msg_t *)buffer,
    (msg_t *)buffer + size,
    (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

<table>
<thead>
<tr>
<th></th>
<th>name</th>
<th>the name of the mailbox variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>buffer</td>
<td>pointer to the mailbox buffer array of msg_t</td>
</tr>
<tr>
<td>in</td>
<td>size</td>
<td>number of msg_t elements in the buffer array</td>
</tr>
</tbody>
</table>

6.11.2.2  MAILBOX_DECL

#define MAILBOX_DECL(
    name,
    buffer,
    size ) mailbox_t name = __MAILBOX_DATA(name, buffer, size)

Static mailbox initializer.

Statically initialized mailboxes require no explicit initialization using chMObjectInit().

Parameters

<table>
<thead>
<tr>
<th></th>
<th>name</th>
<th>the name of the mailbox variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>buffer</td>
<td>pointer to the mailbox buffer array of msg_t</td>
</tr>
<tr>
<td>in</td>
<td>size</td>
<td>number of msg_t elements in the buffer array</td>
</tr>
</tbody>
</table>

6.11.3  Function Documentation
### 6.11 Mailboxes

#### 6.11.3.1 chMBoxObjectInit()

```c
void chMBoxObjectInit (    
    mailbox_t * mbp,    
    msg_t * buf,    
    size_t n )
```

Initializes a `mailbox_t` object.

**Parameters**

<table>
<thead>
<tr>
<th>out</th>
<th>mbp</th>
<th>the pointer to the <code>mailbox_t</code> structure to be initialized</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>buf</td>
<td>pointer to the messages buffer as an array of <code>msg_t</code></td>
</tr>
<tr>
<td>in</td>
<td>n</td>
<td>number of elements in the buffer array</td>
</tr>
</tbody>
</table>

**Function Class:**

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

#### 6.11.3.2 chMBoxReset()

```c
void chMBoxReset (    
    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 `chMBoxResumeX()`.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>mbp</th>
<th>the pointer to an initialized <code>mailbox_t</code> object</th>
</tr>
</thead>
</table>
Function Class:

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

Here is the call graph for this function:

6.11.3.3 chMBResetI()

```c
void chMBResetI ( mailbox_t * mbp )
```

Resets a `mailbox_t` object.

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

Postcondition

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

Parameters

- **in mbp** the pointer to an initialized `mailbox_t` object

Function Class:

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

Here is the call graph for this function:
6.11 Mailboxes

6.11.3.4 chMBPostTimeout()

```c
msg_t chMBPostTimeout (  
    mailbox_t * mbp,  
    msg_t msg,  
    sysinterval_t timeout )
```

Posts a message into a mailbox.

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

**Parameters**

| in | mbp | the pointer to an initialized mailbox_t object |
|    | msg | the message to be posted on the mailbox       |
|    | timeout | the number of ticks before the operation timeouts, the following special values are allowed: |
|    |     | • TIME_IMMEDIATE immediate timeout. |
|    |     | • TIME_INFINITY no timeout. |

**Returns**

The operation status.

**Return values**

| MSG_OK | if a message has been correctly posted. |
| MSG_RESET | if the mailbox has been reset. |
| MSG_TIMEOUT | if the operation has timed out. |

**Function Class:**

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

Here is the call graph for this function:
6.11.3.5  chMBPostTimeoutS()

msg_t chMBPostTimeoutS {
    mailbox_t * mbp,
    msg_t  msg,
    sysinterval_t timeout
}

Posts a message into a mailbox.

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

Parameters

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>mbp</strong></td>
<td>the pointer to an initialized mailbox_t object</td>
<td></td>
</tr>
<tr>
<td><strong>msg</strong></td>
<td>the message to be posted on the mailbox</td>
<td></td>
</tr>
<tr>
<td><strong>timeout</strong></td>
<td>the number of ticks before the operation timeouts, the following special values are allowed:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• TIME_IMMEDIATE immediate timeout.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• TIME_INFINITE no timeout.</td>
<td></td>
</tr>
</tbody>
</table>

Returns

The operation status.

Return values

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MSG_OK</strong></td>
<td>if a message has been correctly posted.</td>
</tr>
<tr>
<td><strong>MSG_RESET</strong></td>
<td>if the mailbox has been reset.</td>
</tr>
<tr>
<td><strong>MSG_TIMEOUT</strong></td>
<td>if the operation has timed out.</td>
</tr>
</tbody>
</table>

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:

![Call Graph]

### 6.11.3.6 chMBPostI()

```c
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** `mbp` the pointer to an initialized `mailbox_t` object
- **in** `msg` the message to be posted on the mailbox

#### Returns

The operation status.

#### Return values

- **MSG_OK** if a message has been correctly posted.
- **MSG_RESET** if the mailbox has been reset.
- **MSG_TIMEOUT** if the 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:

![Call Graph]

6.11.3.7 chMBPostAheadTimeout()

```c
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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>mbp</code></td>
<td>the pointer to an initialized <code>mailbox_t</code> object</td>
</tr>
<tr>
<td><code>msg</code></td>
<td>the message to be posted on the mailbox</td>
</tr>
<tr>
<td><code>timeout</code></td>
<td>the number of ticks before the operation timeouts, the following special values are allowed:</td>
</tr>
<tr>
<td></td>
<td>• <code>TIME_IMMEDIATE</code> immediate timeout.</td>
</tr>
<tr>
<td></td>
<td>• <code>TIME_INFINITE</code> no timeout.</td>
</tr>
</tbody>
</table>

**Returns**

The operation status.

**Return values**

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>MSG_OK</code></td>
<td>if a message has been correctly posted.</td>
</tr>
<tr>
<td><code>MSG_RESET</code></td>
<td>if the mailbox has been reset.</td>
</tr>
<tr>
<td><code>MSG_TIMEOUT</code></td>
<td>if the operation has timed out.</td>
</tr>
</tbody>
</table>
6.11 Mailboxes

Function Class:

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

Here is the call graph for this function:

![Call Graph](image)

6.11.3.8 chMBPostAheadTimeoutS()

```c
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**

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td><code>mbp</code> the pointer to an initialized <code>mailbox_t</code> object</td>
</tr>
<tr>
<td>in</td>
<td><code>msg</code> the message to be posted on the mailbox</td>
</tr>
<tr>
<td>in</td>
<td><code>timeout</code> the number of ticks before the operation timeouts, the following special values are allowed:</td>
</tr>
<tr>
<td></td>
<td>• <code>TIME_IMMEDIATE</code> immediate timeout.</td>
</tr>
<tr>
<td></td>
<td>• <code>TIME_INFINITE</code> no timeout.</td>
</tr>
</tbody>
</table>

**Returns**

The operation status.

**Return values**

- `MSG_OK` if a message has been correctly posted.
- `MSG_RESET` if the mailbox has been reset.
- `MSG_TIMEOUT` if the operation has timed out.
Function Class:

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

Here is the call graph for this function:

![Call Graph](call_graph.png)

### 6.11.3.9 chMBPostAheadI()

```c
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**

<table>
<thead>
<tr>
<th>in</th>
<th>mbp</th>
<th>the pointer to an initialized mailbox_t object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>msg</td>
<td>the message to be posted on the mailbox</td>
</tr>
</tbody>
</table>

**Returns**

The operation status.

**Return values**

- **MSG_OK** if a message has been correctly posted.
- **MSG_RESET** if the mailbox has been reset.
- **MSG_TIMEOUT** if the 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:

6.11.3.10  chMBFetchTimeout()

```c
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

<table>
<thead>
<tr>
<th>in</th>
<th>mbp</th>
<th>the pointer to an initialized mailbox_t object</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>msgp</td>
<td>pointer to a message variable for the received message</td>
</tr>
<tr>
<td>in</td>
<td>timeout</td>
<td>the number of ticks before the operation timeouts, the following special values are allowed:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• TIME_IMMEDIATE immediate timeout.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• TIME_INFINITE no timeout.</td>
</tr>
</tbody>
</table>

Returns

The operation status.

Return values

| MSG_OK  | if a message has been correctly fetched. |
| MSG_RESET | if the mailbox has been reset. |
| MSG_TIMEOUT | if the operation has timed out. |
Function Class:

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

Here is the call graph for this function:

```
chMBFetchTimeout chMBFetchTimeoutS chDbgCheckClassS
chMBGetUsedCountI
chSchRescheduleS
chThdDequeueNextI
chThdEnqueueTimeoutS
```

6.11.3.11 chMBFetchTimeoutS()

```c
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

<table>
<thead>
<tr>
<th>in</th>
<th>mbp</th>
<th>the pointer to an initialized mailbox_t object</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>msgp</td>
<td>pointer to a message variable for the received message</td>
</tr>
<tr>
<td>in</td>
<td>timeout</td>
<td>the number of ticks before the operation timeouts, the following special values are allowed:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• TIME_IMMEDIATE immediate timeout.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• TIME_INFINITE no timeout.</td>
</tr>
</tbody>
</table>

Returns

The operation status.

Return values

<table>
<thead>
<tr>
<th>MSG_OK</th>
<th>if a message has been correctly fetched.</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSG_RESET</td>
<td>if the mailbox has been reset.</td>
</tr>
<tr>
<td>MSG_TIMEOUT</td>
<td>if the operation has timed out.</td>
</tr>
</tbody>
</table>
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:

![Call Graph](image)

### 6.11.3.12 chMBFetchI()

```c
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* `mbp` the pointer to an initialized `mailbox_t` object  
*out* `msgp` pointer to a message variable for the received message

**Returns**

The operation status.

**Return values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSG_OK</td>
<td>if a message has been correctly fetched.</td>
</tr>
<tr>
<td>MSG_RESET</td>
<td>if the mailbox has been reset.</td>
</tr>
<tr>
<td>MSG_TIMEOUT</td>
<td>if the mailbox is empty and a message cannot be fetched.</td>
</tr>
</tbody>
</table>

Function 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:

```
chMBFetchI
chDbgCheckClassI
chMBGetUsedCountI
chThdDequeueNextI
chSysHalt
chThdDoDequeueNextI
```

### 6.11.3.13 chMBGetSizeI()

```c
static size_t chMBGetSizeI ( const mailbox_t * mbp ) [inline], [static]
```

Returns the mailbox buffer size as number of messages.

**Parameters**

- `mbp`: the pointer to an initialized `mailbox_t` 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.

### 6.11.3.14 chMBGetUsedCountI()

```c
static size_t chMBGetUsedCountI ( const mailbox_t * mbp ) [inline], [static]
```

Returns the number of used message slots into a mailbox.

**Parameters**

- `mbp`: the pointer to an initialized `mailbox_t` object
6.11 Mailboxes

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:

![Call Graph](chMBGetUsedCountI chDbgCheckClassI chSysHalt)

### 6.11.3.15 chMBGetFreeCountI()

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

Returns the number of free message slots into a mailbox.

**Parameters**

- **in** `mbp` the pointer to an initialized `mailbox_t` object

**Returns**

The number of empty message slots.

Function Class:

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

Here is the call graph for this function:

![Call Graph](chMBGetFreeCountI chDbgCheckClassI chMBGetSizeI chMBGetUsedCountI chSysHalt)
6.11.3.16  chMBPeekI()

static msg_t chMBPeekI {
   const mailbox_t * mbp ) [inline], [static]

Returns the next message in the queue without removing it.

Precondition

A message must be waiting in the queue for this function to work or it would return garbage. The correct way to use this macro is to use chMBGetUsedCountI() and then use this macro, all within a lock state.

Parameters

| in | mbp | the pointer to an initialized mailbox_t object |

Returns

The next message in queue.

Function Class:

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

Here is the call graph for this function:

```
chMBPeekI -> chDbgCheckClassI -> chSysHalt
```

6.11.3.17  chMBResumeX()

static void chMBResumeX {
   mailbox_t * mbp ) [inline], [static]

Terminates the reset state.
Parameters

| in | mbp | the pointer to an initialized mailbox_t object |

Function Class:

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

6.12.1 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.

6.12.2 Macro Definition Documentation
6.12 Pipes

6.12.2.1 __PIPE_DATA

#define __PIPE_DATA(
    name,
    buffer,
    size )

Value:

{ (uint8_t *)(buffer),
  (uint8_t *)(buffer) + size,
  (uint8_t *)(buffer),
  (uint8_t *)(buffer),
  (size_t)0,
  false,
  NULL,
  NULL,
  __MUTEX_DATA(name.cmtx),
  __MUTEX_DATA(name.wmtx),
  __MUTEX_DATA(name.rmtx),
}

Data part of a static pipe initializer.

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

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>name</th>
<th>the name of the pipe variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>buffer</td>
<td>pointer to the pipe buffer array of uint8_t</td>
</tr>
<tr>
<td>in</td>
<td>size</td>
<td>number of uint8_t elements in the buffer array</td>
</tr>
</tbody>
</table>

6.12.2.2 PIPE_DECL

#define PIPE_DECL(
    name,
    buffer,
    size ) pipe_t name = __PIPE_DATA(name, buffer, size)

Static pipe initializer.

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

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>name</th>
<th>the name of the pipe variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>buffer</td>
<td>pointer to the pipe buffer array of uint8_t</td>
</tr>
<tr>
<td>in</td>
<td>size</td>
<td>number of uint8_t elements in the buffer array</td>
</tr>
</tbody>
</table>

6.12.3 Function Documentation
6.12.3.1 pipe_write()

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

Non-blocking pipe write.

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

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in pp</td>
<td>the pointer to an initialized pipe_t object</td>
</tr>
<tr>
<td>in bp</td>
<td>pointer to the data buffer</td>
</tr>
<tr>
<td>in n</td>
<td>the maximum amount of data to be transferred, the value 0 is reserved</td>
</tr>
</tbody>
</table>

Returns

The number of bytes effectively transferred.

Function Class:

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

6.12.3.2 pipe_read()

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

Non-blocking pipe read.

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

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in pp</td>
<td>the pointer to an initialized pipe_t object</td>
</tr>
<tr>
<td>out bp</td>
<td>pointer to the data buffer</td>
</tr>
<tr>
<td>in n</td>
<td>the maximum amount of data to be transferred, the value 0 is reserved</td>
</tr>
</tbody>
</table>

Returns

The number of bytes effectively transferred.
Function Class:
Not an API, this function is for internal use only.

6.12.3.3 chPipeObjectInit()

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

Initializes a mailbox_t object.

Parameters

| In | pp | the pointer to the pipe_t structure to be initialized |
| In | buf | pointer to the pipe buffer as an array of uint8_t |
| In | n  | 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.

6.12.3.4 chPipeReset()

void chPipeReset (
    pipe_t * pp )

Resets a pipe_t object.

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

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

Parameters

| In | pp | the pointer to an initialized pipe_t object |

Function Class:
Normal API, this function can be invoked by regular system threads but not from within a lock zone.
6.12.3.5 chPipeWriteTimeout()

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

Pipe write with timeout.

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

Parameters

| in | pp | the pointer to an initialized pipe_t object |
| in | bp | pointer to the data buffer |
| in | n  | the number of bytes to be written, the value 0 is reserved |
| in | timeout | the number of ticks before the operation timeouts, the following special values are allowed: |
|    |     | • TIME_IMMEDIATE immediate timeout. |
|    |     | • TIME_INFINITE no timeout. |

Returns

The 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.

6.12.3.6 chPipeReadTimeout()

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

Pipe read with timeout.

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

Parameters

| in | pp | the pointer to an initialized pipe_t object |
| out | bp | pointer to the data buffer |
| in | n  | the number of bytes to be read, the value 0 is reserved |
| in | timeout | the number of ticks before the operation timeouts, the following special values are allowed: |
|    |     | • TIME_IMMEDIATE immediate timeout. |
|    |     | • TIME_INFINITE no timeout. |
Returns

The 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.

6.12.3.7 chPipeGetSize()

\[
\text{static size_t chPipeGetSize (}
\text{ const pipe_t * pp ) [inline], [static]}
\]

Returns the pipe buffer size as number of bytes.

Parameters

| in | pp | the pointer to an initialized pipe_t 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.

6.12.3.8 chPipeGetUsedCount()

\[
\text{static size_t chPipeGetUsedCount (}
\text{ const pipe_t * pp ) [inline], [static]}
\]

Returns the number of used byte slots into a pipe.

Parameters

| in | pp | the pointer to an initialized pipe_t 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.

6.12.3.9 chPipeGetFreeCount()

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

Returns the number of free byte slots into a pipe.

Parameters

| in  | pp | the pointer to an initialized pipe_t object |

Returns
  The number of empty byte slots.

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

Here is the call graph for this function:

6.12.3.10 chPipeResume()

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

Terminates the reset state.
Parameters

| in | pp | the pointer to an initialized pipe_t object |

Function Class:

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

6.13.1 Detailed Description

**Typedefs**

- typedef msg_t(∗ delegate_veneer_t) (va_list ∗ argsp)
  Type of a delegate veneer function.
- typedef msg_t(∗ delegate_fn0_t) (void)
  Type of a delegate function with no parameters.
- typedef msg_t(∗ delegate_fn1_t) (msg_t p1)
  Type of a delegate function with one parameter.
- typedef msg_t(∗ delegate_fn2_t) (msg_t p1, msg_t p2)
  Type of a delegate function with two parameters.
- typedef msg_t(∗ delegate_fn3_t) (msg_t p1, msg_t p2, msg_t p3)
  Type of a delegate function with three parameters.
- typedef msg_t(∗ delegate_fn4_t) (msg_t p1, msg_t p2, msg_t p3, msg_t p4)
  Type of a delegate function with four parameters.

**Functions**

- msg_t __ch_delegate_fn0 (va_list ∗ argsp)
  Veneer for functions with no parameters.
- msg_t __ch_delegate_fn1 (va_list ∗ argsp)
  Veneer for functions with one parameter.
- msg_t __ch_delegate_fn2 (va_list ∗ argsp)
  Veneer for functions with two parameters.
- msg_t __ch_delegate_fn3 (va_list ∗ argsp)
  Veneer for functions with three parameters.
- msg_t __ch_delegate_fn4 (va_list ∗ argsp)
  Veneer for functions with four parameters.
- msg_t chDelegateCallVeneer (thread_t ∗ tp, delegate_veneer_t veneer,...)
  Triggers a function call on a delegate thread.
- void chDelegateDispatch (void)
  Call messages dispatching.
- msg_t chDelegateDispatchTimeout (sysinterval_t timeout)
  Call messages dispatching with timeout.
- static msg_t chDelegateCallDirect0 (thread_t ∗ tp, delegate_fn0_t func)
  Direct call to a function with no parameters.
- static msg_t chDelegateCallDirect1 (thread_t ∗ tp, delegate_fn1_t func, msg_t p1)
  Direct call to a function with one parameter.
- static msg_t chDelegateCallDirect2 (thread_t ∗ tp, delegate_fn2_t func, msg_t p1, msg_t p2)
  Direct call to a function with two parameters.
- static msg_t chDelegateCallDirect3 (thread_t ∗ tp, delegate_fn3_t func, msg_t p1, msg_t p2, msg_t p3)
  Direct call to a function with three parameters.
- static msg_t chDelegateCallDirect4 (thread_t ∗ tp, delegate_fn4_t func, msg_t p1, msg_t p2, msg_t p3, msg_t p4)
  Direct call to a function with four parameters.
6.13 Delegate Threads

6.13.2 Typedef Documentation

6.13.2.1 delegate_veneer_t
typedef msg_t(* delegate_veneer_t) (va_list *argsp)
Type of a delegate veneer function.

6.13.2.2 delegate_fn0_t
typedef msg_t(* delegate_fn0_t) (void)
Type of a delegate function with no parameters.

6.13.2.3 delegate_fn1_t
typedef msg_t(* delegate_fn1_t) (msg_t p1)
Type of a delegate function with one parameter.

6.13.2.4 delegate_fn2_t
typedef msg_t(* delegate_fn2_t) (msg_t p1, msg_t p2)
Type of a delegate function with two parameters.

6.13.2.5 delegate_fn3_t
typedef msg_t(* delegate_fn3_t) (msg_t p1, msg_t p2, msg_t p3)
Type of a delegate function with three parameters.

6.13.2.6 delegate_fn4_t
typedef msg_t(* delegate_fn4_t) (msg_t p1, msg_t p2, msg_t p3, msg_t p4)
Type of a delegate function with four parameters.

6.13.3 Function Documentation

6.13.3.1 __ch_delegate_fn0()
msg_t __ch_delegate_fn0 {
    va_list * argsp
}
Veneer for functions with no parameters.
6.13.3.2 _ch_delegate_fn1()

```c
msg_t _ch_delegate_fn1 {
    va_list * argsp
}
```

Veneer for functions with one parameter.

Parameters

- `in argsp` the list of arguments

Returns

The function return value.

6.13.3.3 _ch_delegate_fn2()

```c
msg_t _ch_delegate_fn2 {
    va_list * argsp
}
```

Veneer for functions with two parameters.

Parameters

- `in argsp` the list of arguments

Returns

The function return value.

6.13.3.4 _ch_delegate_fn3()

```c
msg_t _ch_delegate_fn3 {
    va_list * argsp
}
```
Veneer for functions with three parameters.
Parameters

| in  | argsp | the list of arguments |

Returns

The function return value.

### 6.13.3.5 __ch_delegate_fn4()

```
msg_t __ch_delegate_fn4 ( 
  va_list * argsp )
```

Veneer for functions with four parameters.

Parameters

| in  | argsp | the list of arguments |

Returns

The function return value.

### 6.13.3.6 chDelegateCallVeneer()

```
msg_t chDelegateCallVeneer ( 
  thread_t * tp, 
  delegate_veneer_t veneer, 
  ... )
```

Triggers a function call on a delegate thread.

**Note**

The thread must be executing `chDelegateDispatchTimeout()` in order to have the functions called.

Parameters

| in  | tp      | pointer to the delegate thread |
| in  | veneer  | pointer to the veneer function to be called |
| in  | ...     | variable number of parameters |
Returns

The function return value casted to msg_t. It is garbage for functions returning void.

Here is the call graph for this function:

```
  chDelegateCallVeneer  chMsgSend
```

### 6.13.3.7 chDelegateDispatch()

```c
void chDelegateDispatch ( void )
```

Call messages dispatching.

The function awaits for an incoming call messages and calls the specified functions, then it returns. In case multiple threads are sending messages then the requests are served in priority order.

Function Class:

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

Here is the call graph for this function:

```
  chDelegateDispatch
  chMsgRelease
  chMsgWait
```

### 6.13.3.8 chDelegateDispatchTimeout()

```c
msg_t chDelegateDispatchTimeout ( sysinterval_t timeout )
```

Call messages dispatching with timeout.

The function awaits for an incoming call messages and calls the specified functions, then it returns. In case multiple threads are sending messages then the requests are served in priority order.
Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>timeout</th>
<th>the number of ticks before the operation timeouts, the following special values are allowed:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>• TIME_INFINITE no timeout.</td>
</tr>
</tbody>
</table>

Returns

The function outcome.

Return values

<table>
<thead>
<tr>
<th>MSG_OK</th>
<th>if a function has been called.</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSG_TIMEOUT</td>
<td>if a timeout occurred.</td>
</tr>
</tbody>
</table>

Function Class:

Normal API, 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 chDelegateCallDirect0()

static msg_t chDelegateCallDirect0 ( 
    thread_t * tp, 
    delegate_fn0_t func ) [inline], [static]

Direct call to a function with no parameters.

Note

The return value is assumed to be not larger than a data pointer type. If you need a portable function then use chDelegateCallVeneer() instead.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>tp</th>
<th>pointer to the delegate thread</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>func</td>
<td>pointer to the function to be called</td>
</tr>
</tbody>
</table>
Returns

The function return value as a `msg_t`.

Here is the call graph for this function:

![Call Graph](chart)

6.13.3.10 chDelegateCallDirect1()

```c
static msg_t chDelegateCallDirect1 (thread_t *tp,
                                    delegate_fn1_t func,
                                    msg_t p1) [inline], [static]
```

Direct call to a function with one parameter.

Note

The return value and parameters are assumed to be not larger than a data pointer type. If you need a portable function then use `chDelegateCallVeneer()` instead.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>tp</th>
<th>pointer to the delegate thread</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>func</td>
<td>pointer to the function to be called</td>
</tr>
<tr>
<td>in</td>
<td>p1</td>
<td>parameter 1 passed as a <code>msg_t</code></td>
</tr>
</tbody>
</table>
Returns

The function return value as a `msg_t`.

Here is the call graph for this function:

```
chDelegateCallDirect1
__ch_delegate_fn1
chDelegateCallVeneer
chMsgSend
```

### 6.13.3.11 chDelegateCallDirect2()

```c
static msg_t chDelegateCallDirect2 {
    thread_t * tp,
    delegate_fn2_t func,
    msg_t p1,
    msg_t p2 ) [inline], [static]
```

Direct call to a function with two parameters.

Note

The return value and parameters are assumed to be not larger than a data pointer type. If you need a portable function then use `chDelegateCallVeneer()` instead.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>tp</th>
<th>pointer to the delegate thread</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>func</td>
<td>pointer to the function to be called</td>
</tr>
<tr>
<td>in</td>
<td>p1</td>
<td>parameter 1 passed as a <code>msg_t</code></td>
</tr>
<tr>
<td>in</td>
<td>p2</td>
<td>parameter 2 passed as a <code>msg_t</code></td>
</tr>
</tbody>
</table>
Returns
The function return value as a `msg_t`.

Here is the call graph for this function:

![Call Graph](image)

### 6.13.3.12 chDelegateCallDirect3()

```c
static msg_t chDelegateCallDirect3 (thread_t *tp,
    delegate_fn3_t func,
    msg_t p1,
    msg_t p2,
    msg_t p3) [inline], [static]
```

Direct call to a function with three parameters.

**Note**

The return value and parameters are assumed to be not larger than a data pointer type. If you need a portable function then use `chDelegateCallVeneer()` instead.

**Parameters**

<table>
<thead>
<tr>
<th></th>
<th>tp</th>
<th>pointer to the delegate thread</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>func</td>
<td>pointer to the function to be called</td>
</tr>
<tr>
<td>in</td>
<td>p1</td>
<td>parameter 1 passed as a <code>msg_t</code></td>
</tr>
<tr>
<td>in</td>
<td>p2</td>
<td>parameter 2 passed as a <code>msg_t</code></td>
</tr>
<tr>
<td>in</td>
<td>p3</td>
<td>parameter 3 passed as a <code>msg_t</code></td>
</tr>
</tbody>
</table>
Returns

The function return value as a `msg_t`.

Here is the call graph for this function:

```
chDelegateCallDirect3
__ch_delegate_fn3
chDelegateCallVeneer chMsgSend
```

6.13.3.13 chDelegateCallDirect4()

```c
static msg_t chDelegateCallDirect4 ( thread_t *tp,
     delegate_fn4_t func,
     msg_t p1,
     msg_t p2,
     msg_t p3,
     msg_t p4 ) [inline], [static]
```

Direct call to a function with four parameters.

Note

The return value and parameters are assumed to be not larger than a data pointer type. If you need a portable function then use `chDelegateCallVeneer()` instead.

Parameters

+ **tp** | pointer to the delegate thread
+ **func** | pointer to the function to be called
+ **p1** | parameter 1 passed as `msg_t`
+ **p2** | parameter 2 passed as `msg_t`
+ **p3** | parameter 3 passed as `msg_t`
+ **p4** | parameter 4 passed as `msg_t`
Returns

The function return value as a `msg_t`.

Here is the call graph for this function:
6.14 Jobs Queues

6.14.1 Detailed Description

Macros

- \#define MSG_JOB_NULL ((msg_t)-2)
  
  Dispatcher return code in case of a JOB_NULL has been received.

Typedefs

- typedef struct ch_jobs_queue jobs_queue_t
  
  Type of a jobs queue.
- typedef void(* job_function_t) (void *arg)
  
  Type of a job function.
- typedef struct ch_job_descriptor job_descriptor_t
  
  Type of a job descriptor.

Data Structures

- struct ch_jobs_queue
  
  Type of a jobs queue.
- struct ch_job_descriptor
  
  Type of a job descriptor.

Functions

- static void chJobObjectInit (jobs_queue_t *jqp, size_t jobsn, job_descriptor_t *jobsbuf, msg_t *msgbuf)
  
  Initializes a jobs queue object.
- static job_descriptor_t * chJobGet (jobs_queue_t *jqp)
  
  Allocates a free job object.
- static job_descriptor_t * chJobGetI (jobs_queue_t *jqp)
  
  Allocates a free job object.
- static job_descriptor_t * chJobGetTimeoutS (jobs_queue_t *jqp, sysinterval_t timeout)
  
  Allocates a free job object.
- static job_descriptor_t * chJobGetTimeout (jobs_queue_t *jqp, sysinterval_t timeout)
  
  Allocates a free job object.
- static void chJobPostI (jobs_queue_t *jqp, job_descriptor_t *jp)
  
  Posts a job object.
- static void chJobPostS (jobs_queue_t *jqp, job_descriptor_t *jp)
  
  Posts a job object.
- static void chJobPost (jobs_queue_t *jqp, job_descriptor_t *jp)
  
  Posts a job object.
- static void chJobPostAheadI (jobs_queue_t *jqp, job_descriptor_t *jp)
  
  Posts an high priority job object.
- static void chJobPostAheadS (jobs_queue_t *jqp, job_descriptor_t *jp)
  
  Posts an high priority job object.
- static void chJobPostAhead (jobs_queue_t *jqp, job_descriptor_t *jp)
  
  Posts an high priority job object.
- static msg_t chJobDispatch (jobs_queue_t *jqp)
  
  Waits for a job then executes it.
- static msg_t chJobDispatchTimeout (jobs_queue_t *jqp, sysinterval_t timeout)
  
  Waits for a job then executes it.
6.14 Jobs Queues

6.14.2 Macro Definition Documentation

6.14.2.1 MSG_JOB_NULL

```c
#define MSG_JOB_NULL ((msg_t)-2)
```
Dispatcher return code in case of a JOB_NULL has been received.

6.14.3 Typedef Documentation

6.14.3.1 jobs_queue_t

```c
typedef struct ch_jobs_queue jobs_queue_t
```
Type of a jobs queue.

6.14.3.2 job_function_t

```c
typedef void(* job_function_t) (void *arg)
```
Type of a job function.

6.14.3.3 job_descriptor_t

```c
typedef struct ch_job_descriptor job_descriptor_t
```
Type of a job descriptor.

6.14.4 Function Documentation

6.14.4.1 chJobObjectInit()

```c
static void chJobObjectInit ( 
    jobs_queue_t * jqp, 
    size_t jobsn, 
    job_descriptor_t * jobsbuf, 
    msg_t * msgbuf ) [inline], [static]
```
Initializes a jobs queue object.
### Parameters

<table>
<thead>
<tr>
<th></th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>out</strong></td>
<td><code>jqp</code></td>
<td>pointer to a <code>jobs_queue_t</code> structure</td>
</tr>
<tr>
<td><strong>in</strong></td>
<td><code>jobsn</code></td>
<td>number of jobs available</td>
</tr>
<tr>
<td><strong>in</strong></td>
<td><code>jobsbuf</code></td>
<td>pointer to the buffer of jobs, it must be able to hold <code>jobsn job_description_t</code> structures</td>
</tr>
<tr>
<td><strong>in</strong></td>
<td><code>msgbuf</code></td>
<td>pointer to the buffer of messages, it must be able to hold <code>jobsn msg_t</code> messages</td>
</tr>
</tbody>
</table>

**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:

![Call Graph](call_graph.png)

#### 6.14.4.2 chJobGet()

```c
static job_description_t * chJobGet ( 
    jobs_queue_t * jqp ) [inline], [static]
```

Allocates a free job object.

**Parameters**

| **in** | `jqp` | pointer to a `jobs_queue_t` structure |

**Returns**

The pointer to the allocated job object.
Function Class:

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

Here is the call graph for this function:

![Call Graph]

### 6.14.4.3 chJobGetI()

```c
static job_descriptor_t* chJobGetI (jobs_queue_t *jqp) [inline], [static]
```

Allocates a free job object.

**Parameters**

- **in** `jqp` pointer to a `jobs_queue_t` structure

**Returns**

The pointer to the allocated job object.

**Return values**

- `NULL` if a job object is not immediately available.

Function Class:

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

Here is the call graph for this function:

![Call Graph for chJobGetI]
6.14.4.4  chJobGetTimeoutS()

static job_descriptor_t* chJobGetTimeoutS (  
    jobs_queue_t * jqp, 
    sysinterval_t timeout ) [inline], [static]

Allocates a free job object.

Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>jqp</td>
<td>pointer to a jobs_queue_t structure</td>
</tr>
<tr>
<td>in</td>
<td>timeout</td>
<td>the number of ticks before the operation timeouts, the following special values are allowed:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• TIME_IMMEDIATE immediate timeout.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• TIME_INFINITE no timeout.</td>
</tr>
</tbody>
</table>

Returns

The pointer to the allocated job object.

Return values

<table>
<thead>
<tr>
<th>Return value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NULL</td>
<td>if a job object is not available within the specified timeout.</td>
</tr>
</tbody>
</table>

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:

```
chJobGetTimeoutS  
  |               |               |
  |               | chGuardedPoolAllocTimeoutS |
  |               | chSemWaitTimeoutS |
```

6.14.4.5  chJobGetTimeout()

static job_descriptor_t* chJobGetTimeout (  
    jobs_queue_t * jqp, 
    sysinterval_t timeout ) [inline], [static]

Allocates a free job object.
6.14 Jobs Queues

Parameters

| in | jqp | pointer to a jobs_queue_t structure |
| in | timeout | the number of ticks before the operation timeouts, the following special values are allowed: |
|    |        | • TIME_IMMEDIATE immediate timeout. |
|    |        | • TIME_INFINITE no timeout. |

Returns

The pointer to the allocated job object.

Return values

NULL if a job object is not available within the specified timeout.

Function Class:

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

Here is the call graph for this function:

6.14.4.6 chJobPostI()

static void chJobPostI (jobs_queue_t * jqp, job_descriptor_t * jp) [inline], [static]

Posts a job object.

Note

By design the object can be always immediately posted.

Parameters

| in | jqp | pointer to a jobs_queue_t structure |
| in | jp | pointer to the job object to be posted |
Function Class:

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

Here is the call graph for this function:

![Call Graph]

6.14.4.7 chJobPostS()

```c
static void chJobPostS (jobs_queue_t * jqp, job_descriptor_t * jp) [inline], [static]
```

Posts a job object.

**Note**

By design the object can be always immediately posted.

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in jqp</td>
<td>pointer to a jobs_queue_t structure</td>
</tr>
<tr>
<td>in jp</td>
<td>pointer to the job object to be posted</td>
</tr>
</tbody>
</table>
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:

![Call Graph for chJobPost()](image)

### 6.14.4.8 chJobPost()

```c
static void chJobPost ( 
    jobs_queue_t * jqp, 
    job_descriptor_t * jp ) [inline], [static]
```

Posts a job object.

**Note**

By design the object can be always immediately posted.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>in</strong> jqp</td>
<td>pointer to a jobs_queue_t structure</td>
</tr>
<tr>
<td><strong>in</strong> jp</td>
<td>pointer to the job object to be posted</td>
</tr>
</tbody>
</table>

**Function Class:**

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

Here is the call graph for this function:
6.14.4.9 chJobPostAheadI()

static void chJobPostAheadI (  
    jobs_queue_t * jqp,  
    job_descriptor_t * jp ) [inline], [static]

Posts an high priority job object.

Note

By design the object can be always immediately posted.

Parameters

| in  | jqp | pointer to a jobs_queue_t structure |
| in  | jp  | pointer to the job object to be posted |

Function Class:

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

Here is the call graph for this function:

![Call Graph](image)

6.14.4.10 chJobPostAheadS()

static void chJobPostAheadS (  
    jobs_queue_t * jqp,  
    job_descriptor_t * jp ) [inline], [static]

Posts an high priority job object.

Note

By design the object can be always immediately posted.
6.14 Jobs Queues

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in qjp</td>
<td>pointer to a <code>jobs_queue_t</code> structure</td>
</tr>
<tr>
<td>in jp</td>
<td>pointer to the job object to be posted</td>
</tr>
</tbody>
</table>

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:

```
chJobPostAheadS chMBPostAheadTimeoutS chDbgCheckClassS
chMBGetFreeCountI
chSchRescheduleS
chThdDequeueNextI
chThdEnqueueTimeoutS
```

6.14.4.11 chJobPostAhead()

```c
static void chJobPostAhead (  
   jobs_queue_t * qjp,
   job_descriptor_t * jp ) [inline], [static]
```

Posts an high priority job object.

Note

By design the object can be always immediately posted.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in qjp</td>
<td>pointer to a <code>jobs_queue_t</code> structure</td>
</tr>
<tr>
<td>in jp</td>
<td>pointer to the job object to be posted</td>
</tr>
</tbody>
</table>
Function Class:

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

Here is the call graph for this function:

```
chJobPostAhead  chMBPostAheadTimeout  chMBPostAheadTimeoutS
```

6.14.4.12  chJobDispatch()

```
static msg_t chJobDispatch (jobs_queue_t * jqp ) [inline], [static]
```

Waits for a job then executes it.

Parameters

- **in** `jqp`  pointer to a jobs_queue_t structure

Returns

The function outcome.

Return values

- **MSG_OK** if a job has been executed.
- **MSG_RESET** if the internal mailbox has been reset.
- **MSG_JOB_NULL** if a JOB_NULL has been received.

Here is the call graph for this function:

```
chJobDispatch  chGuardedPoolFreeI  chMBFetchTimeoutS  chSchRescheduleS
chGuardedPoolFree
chMBFetchTimeout
```

ChibiOS/NIL
6.14 Jobs Queues

6.14.4.13 chJobDispatchTimeout()

static msg_t chJobDispatchTimeout (  
  jobs_queue_t * jqp,  
  sysinterval_t timeout ) [inline], [static]

Waits for a job then executes it.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>jqp</th>
<th>pointer to a jobs_queue_t structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>timeout</td>
<td>the number of ticks before the operation timeouts, the following special values are allowed:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• TIME_IMMEDIATE immediate timeout.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• TIME_INFINITE no timeout.</td>
</tr>
</tbody>
</table>

Returns

The function outcome.

Return values

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>MSG_OK</td>
<td>if a job has been executed.</td>
</tr>
<tr>
<td>MSG_TIMEOUT</td>
<td>if a timeout occurred.</td>
</tr>
<tr>
<td>MSG_RESET</td>
<td>if the internal mailbox has been reset.</td>
</tr>
<tr>
<td>MSG_JOB_NULL</td>
<td>if a JOB_NULL has been received.</td>
</tr>
</tbody>
</table>

Here is the call graph for this function:
6.15 Memory Management

6.15.1 Detailed Description

Memory Management services.

Modules

- Core Memory Manager
- Memory Heaps
- Memory Pools
6.16 Core Memory Manager

6.16.1 Detailed Description

Core Memory Manager related APIs and services.

Operation mode

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

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

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

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

Precondition

In order to use the core memory manager APIs the **CH_CFG_USE_MEMCORE** option must be enabled in `chconf.h`.

Note

Compatible with RT and NIL.

Macros

- `#define CH_CFG_MEMCORE_SIZE 0`
  Managed RAM size.
- `#define chCoreAllocAlignedWithOffsetI chCoreAllocFromTopI`
  Allocates a memory block.
- `#define chCoreAllocAlignedWithOffset chCoreAllocFromTop`
  Allocates a memory block.

Typedefs

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

Data Structures

- `typedef struct memcore_t`
  Type of memory core object.
Functions

- void __core_init (void)
  Low level memory manager initialization.
- void * chCoreAllocFromBaseI (size_t size, unsigned align, size_t offset)
  Allocates a memory block starting from the lowest address upward.
- void * chCoreAllocFromTopI (size_t size, unsigned align, size_t offset)
  Allocates a memory block starting from the top address downward.
- void * chCoreAllocFromBase (size_t size, unsigned align, size_t offset)
  Allocates a memory block starting from the lowest address upward.
- void * chCoreAllocFromTop (size_t size, unsigned align, size_t offset)
  Allocates a memory block starting from the top address downward.
- size_t chCoreGetStatusX (void)
  Core memory status.
- static void * chCoreAllocAlignedI (size_t size, unsigned align)
  Allocates a memory block.
- static void * chCoreAllocAligned (size_t size, unsigned align)
  Allocates a memory block.
- static void * chCoreAllocI (size_t size)
  Allocates a memory block.
- static void * chCoreAlloc (size_t size)
  Allocates a memory block.

Variables

- memcore_t ch_memcore
  Memory core descriptor.

6.16.2 Macro Definition Documentation

6.16.2.1 CH_CFG_MEMCORE_SIZE

#define CH_CFG_MEMCORE_SIZE 0

Managed RAM size.

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

Note

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

Requires CH_CFG_USE_MEMCORE.
6.16 Core Memory Manager

6.16.2.2 chCoreAllocAlignedWithOffsetI

#define chCoreAllocAlignedWithOffsetI chCoreAllocFromTopI

Allocates a memory block.

Note
This is a generic form with unspecified allocation position.

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

6.16.2.3 chCoreAllocAlignedWithOffset

#define chCoreAllocAlignedWithOffset chCoreAllocFromTop

Allocates a memory block.

Note
This is a generic form with unspecified allocation position.

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

6.16.3 Typedef Documentation

6.16.3.1 memgetfunc_t

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

Memory get function.

6.16.3.2 memgetfunc2_t

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

Enhanced memory get function.
6.16.4 Function Documentation

6.16.4.1 __core_init()

void __core_init ( 
    void )

Low level memory manager initialization.

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

6.16.4.2 chCoreAllocFromBaseI()

void * chCoreAllocFromBaseI ( 
    size_t size, 
    unsigned align, 
    size_t offset )

Allocates a memory block starting from the lowest address upward.

This function allocates a block of offset + size bytes. The returned pointer has offset bytes before its address and size bytes after.

Parameters

| in  | size  | the size of the block to be allocated. |
| in  | align | desired memory alignment              |
| in  | offset| aligned pointer offset                |

Returns

A pointer to the allocated memory block.

Return values

| NULL  | allocation failed, core memory exhausted. |

Function Class:

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

Here is the call graph for this function:

![Call Graph Diagram]

### 6.16.4.3 chCoreAllocFromTopI()

```c
void * chCoreAllocFromTopI ( 
    size_t size,  
    unsigned align,  
    size_t offset)
```

Allocates a memory block starting from the top address downward.

This function allocates a block of `offset + size` bytes. The returned pointer has `offset` bytes before its address and `size` bytes after.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th><code>size</code></th>
<th>the size of the block to be allocated.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td><code>align</code></td>
<td>desired memory alignment</td>
</tr>
<tr>
<td>in</td>
<td><code>offset</code></td>
<td>aligned pointer offset</td>
</tr>
</tbody>
</table>

**Returns**

A pointer to the allocated memory block.

**Return values**

- `NULL` allocation failed, core memory exhausted.
**Function Class:**

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

Here is the call graph for this function:

![Call Graph](chCoreAllocFromTopI -> chDbgCheckClassI -> chSysHalt)

### 6.16.4.4 chCoreAllocFromBase()

```c
typedef size_t size_t;

void * chCoreAllocFromBase (  
  size_t size,  
  unsigned align,  
  size_t offset )
```

Allocates a memory block starting from the lowest address upward.

This function allocates a block of offset + size bytes. The returned pointer has offset bytes before its address and size bytes after.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>size</th>
<th>the size of the block to be allocated.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>align</td>
<td>desired memory alignment</td>
</tr>
<tr>
<td>in</td>
<td>offset</td>
<td>aligned pointer offset</td>
</tr>
</tbody>
</table>

**Returns**

A pointer to the allocated memory block.

**Return values**

| NULL | allocation failed, core memory exhausted. |
6.16 Core Memory Manager

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

Here is the call graph for this function:

6.16.4.5 chCoreAllocFromTop()

```c
void * chCoreAllocFromTop ( 
    size_t size, 
    unsigned align, 
    size_t offset )
```

Allocates a memory block starting from the top address downward.

This function allocates a block of offset + size bytes. The returned pointer has offset bytes before its address and size bytes after.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>size</th>
<th>the size of the block to be allocated.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>align</td>
<td>desired memory alignment</td>
</tr>
<tr>
<td>in</td>
<td>offset</td>
<td>aligned pointer offset</td>
</tr>
</tbody>
</table>

Returns

A pointer to the allocated memory block.

Return values

| NULL | allocation failed, core memory exhausted. |
Function Class:

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

Here is the call graph for this function:

```
chCoreAllocFromTop -> chCoreAllocFromTopI -> chDbgCheckClassI
```

### 6.16.4.6 chCoreGetStatusX()

```c
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.

### 6.16.4.7 chCoreAllocAlignedI()

```c
static void* chCoreAllocAlignedI(size_t size, unsigned align) [inline], [static]
```

Allocates a memory block.

The allocated block is guaranteed to be properly aligned to the specified alignment.

Note

This is a generic form with unspecified allocation position.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>size</code></td>
<td>the size of the block to be allocated.</td>
</tr>
<tr>
<td><code>align</code></td>
<td>desired memory alignment</td>
</tr>
</tbody>
</table>
Returns

A pointer to the allocated memory block.

Return values

| NULL | allocation failed, core memory exhausted. |

Function Class:

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

6.16.4.8 chCoreAllocAligned()

static void* chCoreAllocAligned ( size_t size, unsigned align ) [inline], [static]

Allocates a memory block.

The allocated block is guaranteed to be properly aligned to the specified alignment.

Note

This is a generic form with unspecified allocation position.

Parameters

| in | size | the size of the block to be allocated |
| in | align | desired memory alignment |

Returns

A pointer to the allocated memory block.

Return values

| NULL | allocation failed, core memory exhausted. |

Function Class:

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

static void* chCoreAllocI (  
    size_t size ) [inline], [static]

Allocates a memory block.

The allocated block is guaranteed to be properly aligned for a pointer data type.

Note
This is a generic form with unspecified allocation position.

Parameters

| in  | size               | the size of the block to be allocated. |

Returns

A pointer to the allocated memory block.

Return values

| NULL | allocation failed, core memory exhausted. |

Function Class:

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

6.16.4.10 chCoreAlloc()

static void* chCoreAlloc (  
    size_t size ) [inline], [static]

Allocates a memory block.

The allocated block is guaranteed to be properly aligned for a pointer data type.

Note
This is a generic form with unspecified allocation position.

Parameters

| in  | size               | the size of the block to be allocated. |
Returns

A pointer to the allocated memory block.

Return values

| NULL         | allocation failed, core memory exhausted. |

Function Class:

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

6.16.5 Variable Documentation

6.16.5.1 ch_memcore

`memcore_t ch_memcore`

Memory core descriptor.
6.17 Memory Heaps

6.17.1 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.

6.17.2 Macro Definition Documentation

6.17.2.1 CH_HEAP_ALIGNMENT

#define CH_HEAP_ALIGNMENT 8U

Minimum alignment used for heap.

Note

Cannot use the sizeof operator in this macro.

6.17.2.2 CH_HEAP_AREA

#define CH_HEAP_AREA(
    name,
    size )

Value:

\texttt{ALIGNED\_VAR(CH\_HEAP\_ALIGNMENT)\
uint8\_t\ name[MEM\_ALIGN\_NEXT((size), CH\_HEAP\_ALIGNMENT)]}

Allocation of an aligned static heap buffer.
6.17.3 Typedef Documentation

6.17.3.1 memory_heap_t

typedef struct memory_heap memory_heap_t

Type of a memory heap.

6.17.3.2 heap_header_t

typedef union heap_header heap_header_t

Type of a memory heap header.

6.17.4 Function Documentation

6.17.4.1 __heap_init()

void __heap_init (  
    void )

Initializes the default heap.

Function Class:

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

6.17.4.2 chHeapObjectInit()

void chHeapObjectInit (  
    memory_heap_t * heapp,  
    void * buf,  
    size_t size )

Initializes a memory heap from a static memory area.

Note

The heap buffer base and size are adjusted if the passed buffer is not aligned to CH_HEAP_ALIGNMENT. This mean that the effective heap size can be less than size.
6.17 Memory Heaps

Parameters

<table>
<thead>
<tr>
<th>out</th>
<th>heapp</th>
<th>pointer to the memory heap descriptor to be initialized</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>buf</td>
<td>heap buffer base</td>
</tr>
<tr>
<td>in</td>
<td>size</td>
<td>heap size</td>
</tr>
</tbody>
</table>

Function Class:

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

6.17.4.3 chHeapAllocAligned()

```c
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

<table>
<thead>
<tr>
<th>in</th>
<th>heapp</th>
<th>pointer to a heap descriptor or NULL in order to access the default heap.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>size</td>
<td>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.</td>
</tr>
<tr>
<td>in</td>
<td>align</td>
<td>desired memory alignment</td>
</tr>
</tbody>
</table>

Returns

A pointer to the aligned allocated block.

Return values

| NULL | if the block cannot be allocated. |

Function Class:

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

6.17.4.4 chHeapFree()

```c
void chHeapFree ( void * p )
```

Frees a previously allocated memory block.
Parameters

| in | p | pointer to the memory block to be freed |

Function Class:

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

6.17.4.5 chHeapStatus()

```c
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 | heapp | pointer to a heap descriptor or NULL in order to access the default heap. |
| in | totalp | pointer to a variable that will receive the total fragmented free space or NULL |
| in | largestp | 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.

6.17.4.6 chHeapAlloc()

```c
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.
6.17 Memory Heaps

Parameters

| in  | heapp | pointer to a heap descriptor or NULL in order to access the default heap. |
| in  | size  | the size of the block to be allocated. Note that the allocated block may be a bit bigger than the requested size for alignment and fragmentation reasons. |

Returns

A pointer to the allocated block.

Return values

- `NULL` if the block cannot be allocated.

Function Class:

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

Here is the call graph for this function:

![Call Graph](chHeapAlloc chHeapAllocAligned)

6.17.4.7 chHeapGetSize()

```c
static size_t chHeapGetSize (const void * p) [inline], [static]
```

Returns the size of an allocated block.

Note

The returned value is the requested size, the real size is the same value aligned to the next `CH_HEAP_ALIGNMENT` multiple.

Parameters

| in  | p | 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.

6.17.5 Variable Documentation

6.17.5.1 default_heap

memory_heap_t default_heap [static]

Default heap descriptor.
6.18 Memory Pools

6.18.1 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 chPoolFree (memory_pool_t *mp, void *objp)
  Releases an object into a memory pool.

- void chPoolAllocI (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 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.

6.18.2 Macro Definition Documentation
6.18 Memory Pools

6.18.2.1 __MEMORYPOOL_DATA

#define __MEMORYPOOL_DATA(
    name,
    size,
    align,
    provider ) {NULL, size, align, provider}

Data part of a static memory pool initializer.

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

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>in</strong> name</td>
<td>the name of the memory pool variable</td>
</tr>
<tr>
<td><strong>in</strong> size</td>
<td>size of the memory pool contained objects</td>
</tr>
<tr>
<td><strong>in</strong> align</td>
<td>required memory alignment</td>
</tr>
<tr>
<td><strong>in</strong> provider</td>
<td>memory provider function for the memory pool</td>
</tr>
</tbody>
</table>

6.18.2.2 MEMORYPOOL_DECL

#define MEMORYPOOL_DECL(
    name,
    size,
    align,
    provider )
memory_pool_t name = __MEMORYPOOL_DATA(name, size, align, provider)

Static memory pool initializer.

Statically initialized memory pools require no explicit initialization using chPoolInit().

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>in</strong> name</td>
<td>the name of the memory pool variable</td>
</tr>
<tr>
<td><strong>in</strong> size</td>
<td>size of the memory pool contained objects</td>
</tr>
<tr>
<td><strong>in</strong> align</td>
<td>required memory alignment</td>
</tr>
<tr>
<td><strong>in</strong> provider</td>
<td>memory provider function for the memory pool or NULL if the pool is not allowed to grow automatically</td>
</tr>
</tbody>
</table>

6.18.2.3 __GUARDEDMEMORYPOOL_DATA

#define __GUARDEDMEMORYPOOL_DATA(
    name,
    size,
    align )
Value:

```c
{ __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**

<table>
<thead>
<tr>
<th>in</th>
<th>name</th>
<th>the name of the memory pool variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>size</td>
<td>size of the memory pool contained objects</td>
</tr>
<tr>
<td>in</td>
<td>align</td>
<td>required memory alignment</td>
</tr>
</tbody>
</table>

### 6.18.2.4 GUARDEDMEMORYPOOL_DECL

```c
#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**

<table>
<thead>
<tr>
<th>in</th>
<th>name</th>
<th>the name of the guarded memory pool variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>size</td>
<td>size of the memory pool contained objects</td>
</tr>
<tr>
<td>in</td>
<td>align</td>
<td>required memory alignment</td>
</tr>
</tbody>
</table>

### 6.18.3 Function Documentation

#### 6.18.3.1 chPoolObjectInitAligned()

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

Initializes an empty memory pool.
6.18 Memory Pools

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>mp</td>
</tr>
<tr>
<td>in</td>
<td>size</td>
</tr>
<tr>
<td>in</td>
<td>align</td>
</tr>
<tr>
<td>in</td>
<td>provider</td>
</tr>
</tbody>
</table>

Function Class:

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

6.18.3.2 chPoolLoadArray()

```c
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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>mp</td>
</tr>
<tr>
<td>in</td>
<td>p</td>
</tr>
<tr>
<td>in</td>
<td>n</td>
</tr>
</tbody>
</table>
Function Class:

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

Here is the call graph for this function:

```
chPoolLoadArray → chPoolAdd → chPoolFree
```

### 6.18.3.3 chPoolAllocI()

```c
void * chPoolAllocI (  
    memory_pool_t * mp )
```

Allocates an object from a memory pool.

**Precondition**

The memory pool must already be initialized.

**Parameters**

- `in` `mp` pointer to a `memory_pool_t` structure

**Returns**

The pointer to the allocated object.

**Return values**

- `NULL` if pool is empty.
Function Class:

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

Here is the call graph for this function:

![Call Graph](image.png)

6.18.3.4 `chPoolAlloc()`

```c
void * chPoolAlloc ( 
    memory_pool_t * mp )
```

Allocates an object from a memory pool.

**Precondition**

The memory pool must already be initialized.

**Parameters**

| in | mp | pointer to a `memory_pool_t` structure |

**Returns**

The pointer to the allocated object.

**Return values**

| `NULL` | if pool is empty. |
Function Class:

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

Here is the call graph for this function:

---

6.18.3.5 **chPoolFreel()**

```c
void chPoolFreel (  
    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** `mp` pointer to a `memory_pool_t` structure
- **in** `objp` the pointer to the object to be released

Function Class:

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

Here is the call graph for this function:
6.18.3.6 chPoolFree()

void chPoolFree (  
    memory_pool_t * mp,  
    void * objp )

Releases an object into a memory pool.

Precondition

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

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>mp</th>
<th>pointer to a memory_pool_t structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>objp</td>
<td>the pointer to the object to be released</td>
</tr>
</tbody>
</table>

Function Class:

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

Here is the call graph for this function:

6.18.3.7 chGuardedPoolObjectInitAligned()

void chGuardedPoolObjectInitAligned (  
    guarded_memory_pool_t * gmp,  
    size_t size,  
    unsigned align )

Initializes an empty guarded memory pool.
218 Module Documentation

Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td><em>gmp</em></td>
</tr>
<tr>
<td>in</td>
<td><em>size</em></td>
</tr>
<tr>
<td>in</td>
<td><em>align</em></td>
</tr>
</tbody>
</table>

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:

```
chGuardedPoolObjectInitAligned  
```

6.18.3.8 chGuardedPoolLoadArray()

```c
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

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td><em>gmp</em></td>
</tr>
<tr>
<td>in</td>
<td><em>p</em></td>
</tr>
<tr>
<td>in</td>
<td><em>n</em></td>
</tr>
</tbody>
</table>
Function Class:

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

Here is the call graph for this function:

6.18.3.9 chGuardedPoolAllocTimeoutS()

```c
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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>gmp</td>
</tr>
<tr>
<td>in</td>
<td>timeout</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Returns**

The pointer to the allocated object.

**Return values**

- null 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:

![Call Graph]

### 6.18.3.10 chGuardedPoolAllocTimeout()

```c
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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>gmp</strong></td>
<td>pointer to a <code>guarded_memory_pool_t</code> structure</td>
</tr>
<tr>
<td><strong>timeout</strong></td>
<td>the number of ticks before the operation timeouts, the following special values are allowed:</td>
</tr>
<tr>
<td></td>
<td>• <code>TIME_IMMEDIATE</code> immediate timeout.</td>
</tr>
<tr>
<td></td>
<td>• <code>TIME_INFINITY</code> no timeout.</td>
</tr>
</tbody>
</table>

**Returns**

The pointer to the allocated object.

**Return values**

- `NULL` 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.
6.18 Memory Pools

Here is the call graph for this function:

![Call Graph](image)

### 6.18.3.11 chGuardedPoolFree()

```c
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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>gmp</code></td>
<td>pointer to a <code>guarded_memory_pool_t</code> structure</td>
</tr>
<tr>
<td><code>objp</code></td>
<td>the pointer to the object to be released</td>
</tr>
</tbody>
</table>

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.
Here is the call graph for this function:

```
chPoolObjectInit
  |       |
  |       |
  |       |
  |       |
  |       |
  |       |
chPoolObjectInitAligned
```

6.18.3.12 chPoolObjectInit()

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

Initializes an empty memory pool.

**Parameters**

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

**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:
6.18 Memory Pools

6.18.3.13 chPoolAdd()

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

Adds an object to a memory pool.

Precondition

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

Note

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

Parameters

| in | mp | pointer to a memory_pool_t structure |
| in | objp | 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:

![Call Graph]

6.18.3.14 chPoolAddI()

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

Adds an object to a memory pool.
Precondition

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

Note

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

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>mp</th>
<th>pointer to a memory_pool_t structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>obj</td>
<td>the pointer to the object to be added</td>
</tr>
</tbody>
</table>

Function 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:

```
chPoolAddI  chPoolFreeI  chDbgCheckClassI
```

6.18.3.15 chGuardedPoolObjectInit()

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

Initializes an empty guarded memory pool.

Parameters

<table>
<thead>
<tr>
<th>out</th>
<th>gmp</th>
<th>pointer to a guarded_memory_pool_t structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>size</td>
<td>the size of the objects contained in this guarded memory pool, the minimum accepted size is the size of a pointer to void.</td>
</tr>
</tbody>
</table>
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:

6.18.3.16  chGuardedPoolGetCounterI()

```
static cnt_t chGuardedPoolGetCounterI ( guarded_memory_pool_t * gmp ) [inline], [static]
```

Gets the count of objects in a guarded memory pool.

**Precondition**

The guarded memory pool must be already been initialized.

**Parameters**

| in | gmp | pointer to a **guarded_memory_pool_t** structure |

**Returns**

The counter of the guard semaphore.

**Function Class:**

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

6.18.3.17  chGuardedPoolAllocI()

```
static void* chGuardedPoolAllocI ( guarded_memory_pool_t * gmp ) [inline], [static]
```

Allocates an object from a guarded memory pool.

**Precondition**

The guarded memory pool must be already been initialized.
Parameters

in  gmp  pointer to a guarded_memory_pool_t structure

Returns

The pointer to the allocated object.

Return values

NULL 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:

6.18.3.18  chGuardedPoolFreeI()

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

Releases an object into a guarded memory pool.

Precondition

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

Parameters

in  gmp  pointer to a guarded_memory_pool_t structure

in  objp  the pointer to the object to be released
Function Class:

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

Here is the call graph for this function:

6.18.3.19 chGuardedPoolFreeS()

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

Releases an object into a guarded memory pool.

Precondition

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

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>gmp</th>
<th>pointer to a guarded_memory_pool_t structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>objp</td>
<td>the pointer to the object to be released</td>
</tr>
</tbody>
</table>
6.18.3.20  chGuardedPoolAdd()  

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

Adds an object to a guarded memory pool.

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

Note  
  This function is just an alias for chGuardedPoolFree() and has been added for clarity.

Parameters  

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>gmp</td>
</tr>
<tr>
<td>in</td>
<td>objp</td>
</tr>
</tbody>
</table>

Function Class:  
Normal API, this function can be invoked by regular system threads but not from within a lock zone.
Here is the call graph for this function:

```
chGuardedPoolAdd
  chGuardedPoolFree
    chGuardedPoolFreeI
    chSchRescheduleS
```

### 6.18.3.21 chGuardedPoolAddI()

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

Adds an object to a guarded memory pool.

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

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

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>gmp</td>
<td>pointer to a <code>guarded_memory_pool_t</code> structure</td>
</tr>
<tr>
<td>in</td>
<td>objp</td>
<td>the pointer to the object to be added</td>
</tr>
</tbody>
</table>
Function 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:

![Call Graph]

6.18.3.22 chGuardedPoolAddS()

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

Adds an object to a guarded memory pool.

Precondition

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

Note

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

Parameters

| in | gmp | pointer to a guarded_memory_pool_t structure |
| in | objp | 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:
6.19 Complex Services

6.19.1 Detailed Description

Modules

- Objects FIFOs
- Objects Caches
- Dynamic Objects Factory
6.20 Objects FIFOs

6.20.1 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.
### 6.20.2 Typedef Documentation

#### 6.20.2.1 objects_fifo_t

typedef struct ch_objects_fifo objects_fifo_t

Type of an objects FIFO.

### 6.20.3 Function Documentation

#### 6.20.3.1 chFifoObjectInitAligned()

```c
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**

<table>
<thead>
<tr>
<th>out</th>
<th>ofp</th>
<th>pointer to a objects_fifo_t structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>objsize</td>
<td>size of objects</td>
</tr>
<tr>
<td>in</td>
<td>objn</td>
<td>number of objects available</td>
</tr>
<tr>
<td>in</td>
<td>objalign</td>
<td>required objects alignment</td>
</tr>
<tr>
<td>in</td>
<td>objbuf</td>
<td>pointer to the buffer of objects, it must be able to hold objn objects of objsize size with objalign alignment</td>
</tr>
<tr>
<td>in</td>
<td>msgbuf</td>
<td>pointer to the buffer of messages, it must be able to hold objn messages</td>
</tr>
</tbody>
</table>
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:

![Call Graph]

6.20.3.2 chFifoObjectInit()

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

Initializes a FIFO object.

Precondition

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

Parameters

<table>
<thead>
<tr>
<th>out</th>
<th>ofp</th>
<th>pointer to a objects_fifo_t structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>objsize</td>
<td>size of objects</td>
</tr>
<tr>
<td>in</td>
<td>objn</td>
<td>number of objects available</td>
</tr>
<tr>
<td>in</td>
<td>objbuf</td>
<td>pointer to the buffer of objects, it must be able to hold objn objects of objsize size</td>
</tr>
<tr>
<td>in</td>
<td>msgbuf</td>
<td>pointer to the buffer of messages, it must be able to hold objn messages</td>
</tr>
</tbody>
</table>
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:

```
chFifoObjectInit chFifoObjectInitAligned
chGuardedPoolLoadArray
chGuardedPoolObjectInitAligned
chMBObjectInit
```

6.20.3.3 chFifoTakeObjectI()

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

Allocates a free object.

Parameters

```
in ofp pointer to a objects_fifo_t structure
```

Returns

The pointer to the allocated object.

Return values

```
NULL if an object is not immediately available.
```
Function Class:

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

Here is the call graph for this function:

```
chFifoTakeObjectI -> chGuardedPoolAllocI -> chPoolAllocI
```

### 6.20.3.4  chFifoTakeObjectTimeoutS()

```c
static void* chFifoTakeObjectTimeoutS (
    objects_fifo_t * ofp,
    sysinterval_t timeout ) [inline], [static]
```

Allocates a free object.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>ofp</th>
<th>pointer to a objects_fifo_t structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>timeout</td>
<td>the number of ticks before the operation timeouts, the following special values are allowed:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• TIME_IMMEDIATE immediate timeout.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• TIME_INFINITE no timeout.</td>
</tr>
</tbody>
</table>

**Returns**

The pointer to the allocated object.

**Return values**

| NULL | 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:

![Call Graph](image)

6.20.3.5 chFifoTakeObjectTimeout()

```c
static void* chFifoTakeObjectTimeout (
    objects_fifo_t * ofp,
    sysinterval_t timeout ) [inline], [static]
```

Allocates a free object.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in <code>ofp</code></td>
<td>pointer to a <code>objects_fifo_t</code> structure</td>
</tr>
<tr>
<td>in <code>timeout</code></td>
<td>the number of ticks before the operation timeouts, the following special values are allowed:</td>
</tr>
<tr>
<td></td>
<td>• <code>TIME_IMMEDIATE</code> immediate timeout.</td>
</tr>
<tr>
<td></td>
<td>• <code>TIME_INFINITE</code> no timeout.</td>
</tr>
</tbody>
</table>

**Returns**

The pointer to the allocated object.

**Return values**

- `NULL` 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:

![Call Graph]

### 6.20.3.6 chFifoReturnObjectI()

```c
static void chFifoReturnObjectI ( 
    objects_fifo_t * ofp,
    void * objp ) [inline], [static]
```

Releases a fetched object.

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>ofp</td>
</tr>
<tr>
<td>in</td>
<td>obj</td>
</tr>
</tbody>
</table>

Function 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:

![Call Graph]
6.20.3.7  chFifoReturnObjectS()

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

Releases a fetched object.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in ofp</td>
<td>pointer to a objects_fifo_t structure</td>
</tr>
<tr>
<td>in objp</td>
<td>pointer to the object to be released</td>
</tr>
</tbody>
</table>

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:

![Call Graph](call_graph.png)

6.20.3.8  chFifoReturnObject()

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

Releases a fetched object.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in ofp</td>
<td>pointer to a objects_fifo_t structure</td>
</tr>
<tr>
<td>in objp</td>
<td>pointer to the object to be released</td>
</tr>
</tbody>
</table>
Function Class:

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

Here is the call graph for this function:

```
chFifoReturnObject -> chGuardedPoolFree
     |                        | chSchRescheduleS
```

6.20.3.9 chFifoSendObjectI()

```c
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  | ofp         | pointer to a objects_fifo_t structure |
| in  | objp        | 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:

```
chFifoSendObjectI -> chMBPost
     |                        | chDbgCheckClass
     |                        | chMBGetFreeCount
     |                        | chThdDequeueNext
```

ChibiOS/NIL
6.20.3.10  chFifoSendObjectS()

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

Posts an object.

Note
By design the object can be always immediately posted.

Parameters

| in | ofp | pointer to a objects_fifo_t structure |
| in | objp | 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:

![Call Graph](call_graph.png)

6.20.3.11  chFifoSendObject()

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

Posts an object.

Note
By design the object can be always immediately posted.
Parameters

| in | ofp | pointer to a objects_fifo_t structure |
| in | objp | 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:

![Call Graph](image)

6.20.3.12 chFifoSendObjectAheadI()

```c
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 | ofp | pointer to a objects_fifo_t structure |
| in | objp | 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:

6.20.3.13  `chFifoSendObjectAheadS()`

```c
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

<table>
<thead>
<tr>
<th><strong>in</strong></th>
<th><strong>ofp</strong></th>
<th>pointer to a <code>objects_fifo_t</code> structure</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>in</strong></td>
<td><strong>objp</strong></td>
<td>pointer to the object to be posted</td>
</tr>
</tbody>
</table>

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:
6.20.3.14 chFifoSendObjectAhead()

static void chFifoSendObjectAhead (  
  objects_fifo_t ∗ ofp,  
  void ∗ objp ) [inline], [static]

Posts an high priority object.

Note

By design the object can be always immediately posted.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>ofp</th>
<th>pointer to a objects_fifo_t structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>objp</td>
<td>pointer to the object to be released</td>
</tr>
</tbody>
</table>

Function Class:

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

Here is the call graph for this function:

![Call Graph]

6.20.3.15 chFifoReceiveObjectI()

static msg_t chFifoReceiveObjectI (  
  objects_fifo_t ∗ ofp,  
  void ∗ ∗ objpp ) [inline], [static]

Fetches an object.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>ofp</th>
<th>pointer to a objects_fifo_t structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>objpp</td>
<td>pointer to the fetched object reference</td>
</tr>
</tbody>
</table>
Returns

The operation status.

Return values

| MSG_OK          | if an object has been correctly fetched. |
| MSG_TIMEOUT     | 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:

![Call Graph]

6.20.3.16 chFifoReceiveObjectTimeoutS()

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

Fetched an object.

Parameters

| in  | ofp        | pointer to a objects_fifo_t structure |
|     | objpp      | pointer to the fetched object reference |
| in  | timeout    | the number of ticks before the operation timeouts, the following special values are allowed: |
|     |            | • TIME_IMMEDIATE immediate timeout. |
|     |            | • TIME_INFINITE no timeout. |

Returns

The operation status.
Return values

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSG_OK</td>
<td>if an object has been correctly fetched.</td>
</tr>
<tr>
<td>MSG_TIMEOUT</td>
<td>if the operation has timed out.</td>
</tr>
</tbody>
</table>

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:

![Call Graph](image)

### 6.20.3.17 chFifoReceiveObjectTimeout()

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

Fetched an object.

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>*ofp</td>
<td>pointer to a <code>objects_fifo_t</code> structure</td>
</tr>
<tr>
<td>in</td>
<td>*objpp</td>
<td>pointer to the fetched object reference</td>
</tr>
</tbody>
</table>
| in    | timeout| the number of ticks before the operation timeouts, the following special values are allowed:  
|       |       | • `TIME_IMMEDIATE` immediate timeout.             |
|       |       | • `TIME_INFINITE` no timeout.                     |

**Returns**

The operation status.
Return values

<table>
<thead>
<tr>
<th>MSG_OK</th>
<th>if an object has been correctly fetched.</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSG_TIMEOUT</td>
<td>if the operation has timed out.</td>
</tr>
</tbody>
</table>

Function Class:

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

Here is the call graph for this function:
6.21 Objects Caches

6.21.1 Detailed Description

Cached objects flags

- `#define OC_FLAG_INLRU 0x00000001U`
- `#define OC_FLAG_INHASH 0x00000002U`
- `#define OC_FLAG_SHARED 0x00000004U`
- `#define OC_FLAG_NOTSYNC 0x00000008U`
- `#define OC_FLAG_LAZYWRITE 0x00000010U`
- `#define OC_FLAG_FORGET 0x00000020U`

Typedefs

- `typedef uint32_t oc_flags_t`
  
  Flags of cached objects.

- `typedef struct ch_oc_hash_header oc_hash_header_t`
  
  Type of a hash element header.

- `typedef struct ch_oc_lru_header oc_lru_header_t`
  
  Type of an LRU element header.

- `typedef struct ch_oc_object oc_object_t`
  
  Type of a cached object.

- `typedef struct ch_objects_cache objects_cache_t`
  
  Type of a cache object.

- `typedef bool(∗oc_readf)(objects_cache_t ∗ocp, oc_object_t ∗objp, bool async)`
  
  Object read function.

- `typedef bool(∗oc_writef)(objects_cache_t ∗ocp, oc_object_t ∗objp, bool async)`
  
  Object write function.

Data Structures

- `struct ch_oc_hash_header`
  
  Structure representing a hash table element.

- `struct ch_oc_lru_header`
  
  Structure representing a hash table element.

- `struct ch_oc_object`
  
  Structure representing a cached object.

- `struct ch_objects_cache`
  
  Structure representing a cache object.
Functions

- static oc_object_t * hash_get_s (objects_cache_t *ocp, uint32_t group, uint32_t key)
  
  Returns an object pointer from the cache, if present.

- static oc_object_t * lru_get_last_s (objects_cache_t *ocp)
  
  Gets the least recently used object buffer from the LRU list.

- void chCacheObjectInit (objects_cache_t *ocp, ucnt_t hashn, oc_hash_header_t *hashp, ucnt_t objn, size_t objsz, void *objvp, oc_readf_t readf, oc_writef_t writef)
  
  Initializes a objects_cache_t object.

- oc_object_t * chCacheGetObject (objects_cache_t *ocp, uint32_t group, uint32_t key)
  
  Retrieves an object from the cache.

- void chCacheReleaseObjectI (objects_cache_t *ocp, oc_object_t *objp)
  
  Releases an object into the cache.

- bool chCacheReadObject (objects_cache_t *ocp, oc_object_t *objp, bool async)
  
  Reads object data from the storage.

- bool chCacheWriteObject (objects_cache_t *ocp, oc_object_t *objp, bool async)
  
  Writes the object data back to storage.

- static void chCacheReleaseObject (objects_cache_t *ocp, oc_object_t *objp)
  
  Releases an object into the cache.

6.21.2 Typedef Documentation

6.21.2.1 oc_flags_t

typedef uint32_t oc_flags_t

Flags of cached objects.

6.21.2.2 oc_hash_header_t

typedef struct ch_oc_hash_header oc_hash_header_t

Type of an hash element header.

6.21.2.3 oc_lru_header_t

typedef struct ch_oc_lru_header oc_lru_header_t

Type of an LRU element header.
6.21 Objects Caches

### 6.21.2.4 oc_object_t

typedef struct ch_oc_object oc_object_t

Type of a cached object.

### 6.21.2.5 objects_cache_t

typedef struct ch_objects_cache objects_cache_t

Type of a cache object.

### 6.21.2.6 oc_readf_t

typedef bool(* oc_readf_t) (objects_cache_t *ocp, oc_object_t *objp, bool async)

Object read function.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in <code>ocp</code></td>
<td>pointer to the objects_cache_t structure</td>
</tr>
<tr>
<td>in <code>async</code></td>
<td>requests an asynchronous operation if supported, the function is then responsible for releasing the object</td>
</tr>
</tbody>
</table>

### 6.21.2.7 oc_writef_t

typedef bool(* oc_writef_t) (objects_cache_t *ocp, oc_object_t *objp, bool async)

Object write function.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in <code>ocp</code></td>
<td>pointer to the objects_cache_t structure</td>
</tr>
<tr>
<td>in <code>async</code></td>
<td>requests an asynchronous operation if supported, the function is then responsible for releasing the object</td>
</tr>
</tbody>
</table>

### 6.21.3 Function Documentation
6.21.3.1 hash_get_s()

```c
static oc_object_t * hash_get_s (  
    objects_cache_t * ocp,  
    uint32_t group,  
    uint32_t key ) [static]
```

Returns an object pointer from the cache, if present.

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ocp</td>
<td>objects_cache_t *</td>
<td>pointer to the objects_cache_t structure to be initialized</td>
</tr>
<tr>
<td>group</td>
<td>uint32_t</td>
<td>object group identifier</td>
</tr>
<tr>
<td>key</td>
<td>uint32_t</td>
<td>object identifier within the group initialized</td>
</tr>
</tbody>
</table>

Returns

The pointer to the retrieved object.

Return values

- `NULL` if the object is not in cache.

Function Class:

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

6.21.3.2 lru_get_last_s()

```c
static oc_object_t * lru_get_last_s (  
    objects_cache_t * ocp ) [static]
```

Gets the least recently used object buffer from the LRU list.

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ocp</td>
<td>objects_cache_t *</td>
<td>pointer to the objects_cache_t structure to be initialized</td>
</tr>
</tbody>
</table>

Returns

The pointer to the retrieved object.

Function Class:

Not an API, this function is for internal use only.
6.21.3.3 chCacheObjectInit()

```c
void chCacheObjectInit ( 
    objects_cache_t * ocp,
    ucnt_t hashn,
    oc_hash_header_t * hashp,
    ucnt_t objn,
    size_t objsz,
    void * objvp,
    oc_readf_t readf,
    oc_writef_t writef )
```

Initializes a objects_cache_t object.

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td><code>ocp</code> pointer to the objects_cache_t structure to be initialized</td>
</tr>
<tr>
<td>in</td>
<td><code>hashn</code> number of elements in the hash table array, must be a power of two and not lower than <code>objn</code></td>
</tr>
<tr>
<td>in</td>
<td><code>hashp</code> pointer to the hash table as an array of oc_hash_header_t</td>
</tr>
<tr>
<td>in</td>
<td><code>objn</code> number of elements in the objects table array</td>
</tr>
<tr>
<td>in</td>
<td><code>objsz</code> size of elements in the objects table array, the minimum value is sizeof (oc_object_t).</td>
</tr>
<tr>
<td>in</td>
<td><code>objvp</code> pointer to the hash objects as an array of structures starting with an oc_object_t</td>
</tr>
<tr>
<td>in</td>
<td><code>readf</code> pointer to an object reader function</td>
</tr>
<tr>
<td>in</td>
<td><code>writef</code> pointer to an object writer function</td>
</tr>
</tbody>
</table>

**Function Class:**

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

6.21.3.4 chCacheGetObject()

```c
oc_object_t * chCacheGetObject ( 
    objects_cache_t * ocp,
    uint32_t group,
    uint32_t key )
```

Retrieves an object from the cache.

**Note**

If the object is not in cache then the returned object is marked as OC_FLAG_NOTSYNC meaning that its data contains garbage and must be initialized.

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td><code>ocp</code> pointer to the objects_cache_t structure</td>
</tr>
<tr>
<td>in</td>
<td><code>group</code> object group identifier</td>
</tr>
<tr>
<td>in</td>
<td><code>key</code> object identifier within the group</td>
</tr>
</tbody>
</table>
Returns

The pointer to the retrieved object.

Function Class:

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

Here is the call graph for this function:

6.21.3.5 chCacheReleaseObject()

void chCacheReleaseObjectI (  
    objects_cache_t * ocp,  
    oc_object_t * objp )

Releases an object into the cache.

Note

This function gives a meaning to the following flags:

• OC_FLAG_INLRU must be cleared.
• OC_FLAG_INHASH must be set.
• OC_FLAG_SHARED must be cleared.
• OC_FLAG_NOTSYNC invalidates the object and queues it on the LRU tail.
• OC_FLAG_LAZYWRITE is ignored and kept, a write will occur when the object is removed from the LRU list (lazy write).

Parameters

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td></td>
<td>in</td>
</tr>
<tr>
<td>ocp</td>
<td>pointer to the objects_cache_t structure</td>
<td>objp</td>
</tr>
</tbody>
</table>

Function Class:

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

```c
bool chCacheReadObject(
    objects_cache_t *ocp,
    oc_object_t *objp,
    bool async )
```

Reads object data from the storage.

Note

In case of asynchronous operation an error condition is not reported by this function.

Parameters

<table>
<thead>
<tr>
<th>param</th>
<th>type</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ocp</td>
<td>pointer to the objects_cache_t structure</td>
<td></td>
</tr>
<tr>
<td>objp</td>
<td>pointer to the oc_object_t structure</td>
<td></td>
</tr>
<tr>
<td>async</td>
<td>requests an asynchronous operation if supported, the function is then responsible for releasing the object</td>
<td></td>
</tr>
</tbody>
</table>

Returns

The operation status. In case of asynchronous operation false is always returned.

Return values

<table>
<thead>
<tr>
<th>value</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>false</td>
<td>if the operation succeeded.</td>
</tr>
<tr>
<td>true</td>
<td>if the synchronous read operation failed.</td>
</tr>
</tbody>
</table>

Function Class:

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

6.21.3.7 chCacheWriteObject() 

```c
bool chCacheWriteObject(
    objects_cache_t *ocp,
    oc_object_t *objp,
    bool async )
```

Writes the object data back to storage.

Note

In case of asynchronous operation an error condition is not reported by this function.
6.21.3.8 chCacheReleaseObject()

Releases an object into the cache.

Note

This function gives a meaning to the following flags:

- **OC_FLAG_INLRU** must be cleared.
- **OC_FLAG_INHASH** must be set.
- **OC_FLAG_SHARED** must be cleared.
- **OC_FLAG_NOTSYNC** invalidates the object and queues it on the LRU tail.
- **OC_FLAG_LAZYWRITE** is ignored and kept, a write will occur when the object is removed from the LRU list (lazy write).

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ocp</code></td>
<td>pointer to the <code>objects_cache_t</code> structure</td>
</tr>
<tr>
<td><code>objp</code></td>
<td>pointer to the <code>oc_object_t</code> structure</td>
</tr>
</tbody>
</table>
6.21 Objects Caches

Function Class:

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

Here is the call graph for this function:
6.22 Dynamic Objects Factory

6.22.1 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_GENERICBuffers TRUE
  Enables factory for generic buffers.
- #define CH_CFG_FACTORY_SEMAPHORES TRUE
  Enables factory for semaphores.
- #define CH_CFG_FACTORY_SEMAPHORES FALSE
  Enables factory for semaphores.
- #define CH_CFG_FACTORY_MAILBOXES TRUE
  Enables factory for mailboxes.
- #define CH_CFG_FACTORY_MAILBOXES FALSE
  Enables factory for mailboxes.
- #define CH_CFG_FACTORY_OBJ_FIFOS TRUE
  Enables factory for objects FIFOs.
- #define CH_CFG_FACTORY_OBJ_FIFOS FALSE
  Enables factory for objects FIFOs.
- #define CH_CFG_FACTORY_PIPES TRUE
  Enables factory for Pipes.
- #define CH_CFG_FACTORY_PIPES FALSE
  Enables factory for Pipes.
6.22 Dynamic Objects Factory

**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 * chFactoryFindObjectByFIFO (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.
6.22 Dynamic Objects Factory

- 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.

6.22.2 Macro Definition Documentation

6.22.2.1 CH_CFG_FACTORY_MAX_NAMES_LENGTH

#define CH_CFG_FACTORY_MAX_NAMES_LENGTH 8

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

6.22.2.2 CH_CFG_FACTORY_OBJECTS_REGISTRY

#define CH_CFG_FACTORY_OBJECTS_REGISTRY TRUE

Enables the registry of generic objects.

6.22.2.3 CH_CFG_FACTORY_GENERIC_BUFFERS

#define CH_CFG_FACTORY_GENERIC_BUFFERS TRUE

Enables factory for generic buffers.

6.22.2.4 CH_CFG_FACTORY_SEMAPHORES [1/2]

#define CH_CFG_FACTORY_SEMAPHORES TRUE

Enables factory for semaphores.
6.22.2.5  CH_CFG_FACTORY_SEMAPHORES [2/2]

#define CH_CFG_FACTORY_SEMAPHORES FALSE

Enables factory for semaphores.

6.22.2.6  CH_CFG_FACTORY_MAILBOXES [1/2]

#define CH_CFG_FACTORY_MAILBOXES TRUE

Enables factory for mailboxes.

6.22.2.7  CH_CFG_FACTORY_MAILBOXES [2/2]

#define CH_CFG_FACTORY_MAILBOXES FALSE

Enables factory for mailboxes.

6.22.2.8  CH_CFG_FACTORY_OBJ_FIFOS [1/3]

#define CH_CFG_FACTORY_OBJ_FIFOS TRUE

Enables factory for objects FIFOs.

6.22.2.9  CH_CFG_FACTORY_OBJ_FIFOS [2/3]

#define CH_CFG_FACTORY_OBJ_FIFOS TRUE

Enables factory for objects FIFOs.

6.22.2.10  CH_CFG_FACTORY_OBJ_FIFOS [3/3]

#define CH_CFG_FACTORY_OBJ_FIFOS FALSE

Enables factory for objects FIFOs.
6.22 Dynamic Objects Factory

6.22.2.11 CH_CFG_FACTORY_PIPES [1/2]

#define CH_CFG_FACTORY_PIPES TRUE

Enables factory for Pipes.

6.22.2.12 CH_CFG_FACTORY_PIPES [2/2]

#define CH_CFG_FACTORY_PIPES FALSE

Enables factory for Pipes.

6.22.3 Typedef Documentation

6.22.3.1 dyn_element_t

typedef struct ch_dyn_element dyn_element_t

Type of a dynamic object list element.

6.22.3.2 dyn_list_t

typedef struct ch_dyn_list dyn_list_t

Type of a dynamic object list.

6.22.3.3 registered_object_t

typedef struct ch_registered_static_object registered_object_t

Type of a registered object.

6.22.3.4 dyn_buffer_t

typedef struct ch_dyn_object dyn_buffer_t

Type of a dynamic buffer object.
6.22.3.5 dyn_semaphore_t

typedef struct ch_dyn_semaphore dyn_semaphore_t

Type of a dynamic semaphore.

6.22.3.6 dyn_mailbox_t

typedef struct ch_dyn_mailbox dyn_mailbox_t

Type of a dynamic buffer object.

6.22.3.7 dyn_objects_fifo_t

typedef struct ch_dyn_objects_fifo dyn_objects_fifo_t

Type of a dynamic buffer object.

6.22.3.8 dyn_pipe_t

typedef struct ch_dyn_pipe dyn_pipe_t

Type of a dynamic pipe object.

6.22.3.9 objects_factory_t

typedef struct ch_objects_factory objects_factory_t

Type of the factory main object.

6.22.4 Function Documentation
6.22.4.1 __factory_init()

```c
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.

6.22.4.2 chFactoryRegisterObject()

```c
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

<table>
<thead>
<tr>
<th>Type</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>name</td>
<td>name to be assigned to the registered object</td>
</tr>
<tr>
<td>in</td>
<td>objp</td>
<td>pointer to the object to be registered</td>
</tr>
</tbody>
</table>

Returns

The reference to the registered object.

Return values

- **NULL** 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.

6.22.4.3 chFactoryFindObject()

```c
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 | name | name of the registered object |

Returns

The reference to the found registered object.

Return values

| NULL | 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.

6.22.4.4 chFactoryFindObjectByPointer()

`registered_object_t * chFactoryFindObjectByPointer ( void * objp )`

Retrieves a registered object by pointer.

Postcondition

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

Parameters

| in | obj | pointer to the object to be retrieved |

Returns

The reference to the found registered object.

Return values

| NULL | if a registered object with the specified pointer does not exist. |
Function Class:
Normal API, this function can be invoked by regular system threads but not from within a lock zone.

### 6.22.4.5 chFactoryReleaseObject()

```c
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**

<table>
<thead>
<tr>
<th>in</th>
<th>rop</th>
<th>registered object reference</th>
</tr>
</thead>
</table>

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

### 6.22.4.6 chFactoryCreateBuffer()

```c
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**

<table>
<thead>
<tr>
<th>in</th>
<th>name</th>
<th>name to be assigned to the new dynamic buffer object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>size</td>
<td>payload size of the dynamic buffer object to be created</td>
</tr>
</tbody>
</table>
Returns

The reference to the created dynamic buffer object.

Return values

| NULL  | 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.

6.22.4.7 chFactoryFindBuffer()

dyn_buffer_t * chFactoryFindBuffer (const char * name)

Retrieves a dynamic buffer object.

Postcondition

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

Parameters

| in   | name | name of the dynamic buffer object |

Returns

The reference to the found dynamic buffer object.

Return values

| NULL  | if a dynamic buffer object with the specified name does not exist. |

Function Class:

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

6.22.4.8 chFactoryReleaseBuffer()

void chFactoryReleaseBuffer (dyn_buffer_t * dbp)
Releases a dynamic buffer object.

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

| in  | dbp          | dynamic buffer object reference |

Function Class:

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

### 6.22.4.9 chFactoryCreateSemaphore()

```c
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  | name      | name to be assigned to the new dynamic semaphore object |
| in  | n         | dynamic semaphore object counter initialization value |

Returns

The reference to the created dynamic semaphore object.

Return values

| NULL | 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.

### 6.22.4.10 chFactoryFindSemaphore()

```c
dyn_semaphore_t * chFactoryFindSemaphore ( 
    const char * name )
```

Retrieves a dynamic semaphore object.
Postcondition

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

Parameters

| in | name | name of the dynamic semaphore object |

Returns

The reference to the found dynamic semaphore object.

Return values

| NULL | 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.

6.22.4.11 chFactoryReleaseSemaphore()

```c
void chFactoryReleaseSemaphore ( dyn_semaphore_t * dsp )
```

Releases a dynamic semaphore object.

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

Parameters

| in | dsp | dynamic semaphore object reference |

Function Class:

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

6.22.4.12 chFactoryCreateMailbox()

```c
dyn_mailbox_t * chFactoryCreateMailbox ( const char * name, size_t n )
```

Creates a dynamic mailbox object.
Module Documentation

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 | name       | name to be assigned to the new dynamic mailbox object |
|    | in | n          | mailbox buffer size as number of messages |

Returns

The reference to the created dynamic mailbox object.

Return values

| NULL | if the dynamic mailbox object cannot be allocated or a dynamic mailbox object with the same name exists. |

Function Class:

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

6.22.4.13 chFactoryFindMailbox()

dyn_mailbox_t * chFactoryFindMailbox (  
  const char * name )  

Retrieves a dynamic mailbox object.

Postcondition

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

Parameters

| in | name       | name of the dynamic mailbox object |

Returns

The reference to the found dynamic mailbox object.

Return values

| NULL | if a dynamic mailbox object with the specified name does not exist. |
6.22 Dynamic Objects Factory

Function Class:

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

6.22.4.14  chFactoryReleaseMailbox()

```c
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 | dmp | dynamic mailbox object reference |

Function Class:

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

6.22.4.15  chFactoryCreateObjectsFIFO()

```c
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 | name | name to be assigned to the new dynamic "objects FIFO" object |
| in | objsize | size of objects |
| in | objn | number of objects available |
| in | objalign | required objects alignment |
Returns

The reference to the created dynamic "objects FIFO" object.

Return values

| NULL  | if the dynamic "objects FIFO" object cannot be allocated or a dynamic "objects FIFO" object with the same name exists. |

Function Class:

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

6.22.4.16 chFactoryFindObjectsFIFO()

dyn_objects_fifo_t * chFactoryFindObjectsFIFO ( const char * name )

Retrieves a dynamic "objects FIFO" object.

Postcondition

A reference to the dynamic "objects FIFO" object is returned with the reference counter increased by one.

Parameters

| in   | name | name of the dynamic "objects FIFO" object |

Returns

The reference to the found dynamic "objects FIFO" object.

Return values

| NULL  | 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.

6.22.4.17 chFactoryReleaseObjectsFIFO()

void chFactoryReleaseObjectsFIFO ( dyn_objects_fifo_t * dof )
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  | dof | 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.

### 6.22.4.18 chFactoryCreatePipe()

```c
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  | name | name to be assigned to the new dynamic pipe object |
| in  | size | pipe buffer size |

Returns

The reference to the created dynamic pipe object.

Return values

| NULL | 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.
6.22.4.19  chFactoryFindPipe()

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

Retrieves a dynamic pipe object.

**Postcondition**

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

**Parameters**

| in  | name  | name of the pipe object |

**Returns**

The reference to the found dynamic pipe object.

**Return values**

| NULL | 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.

6.22.4.20  chFactoryReleasePipe()

```
void chFactoryReleasePipe (  
    dyn_pipe_t * dpp  
)
```

Releases a dynamic pipe object.

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

**Parameters**

| in  | dpp   | dynamic pipe object reference |

**Function Class:**

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

6.22.4.21  chFactoryDuplicateReference()

```
static dyn_element_t* chFactoryDuplicateReference (  
    dyn_element_t * dep ) [inline], [static]
```

Duplicates an object reference.

**Note**

This function can be used on any kind of dynamic object.

**Parameters**

| in   | dep | pointer to the element field of the object |

**Returns**

The duplicated object reference.

**Function Class:**

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

6.22.4.22  chFactoryGetObject()

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

Returns the pointer to the inner registered object.

**Parameters**

| in   | rop | registered object reference |

**Returns**

The pointer to the registered object.

**Function Class:**

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

static size_t chFactoryGetBufferSize ( 
    dyn_buffer_t * dbp ) [inline], [static]

Returns the size of a generic dynamic buffer object.
6.22 Dynamic Objects Factory

Parameters

| in  | dbp | 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:

```
chFactoryGetBufferSize → chHeapGetSize
```

6.22.4.24 chFactoryGetBuffer()

```
static uint8_t* chFactoryGetBuffer ( dyn_buffer_t* dbp ) [inline], [static]
```

Returns the pointer to the inner buffer.

Parameters

| in  | dbp | 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.
6.22.4.25 chFactoryGetSemaphore()

static semaphore_t * chFactoryGetSemaphore ( 
    dyn_semaphore_t * dsp ) [inline], [static]

Returns the pointer to the inner semaphore.

Parameters

| in | dsp | 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.

6.22.4.26 chFactoryGetMailbox()

static mailbox_t * chFactoryGetMailbox ( 
    dyn_mailbox_t * dmp ) [inline], [static]

Returns the pointer to the inner mailbox.

Parameters

| in | dmp | 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.

6.22.4.27 chFactoryGetObjectsFIFO()

static objects_fifo_t * chFactoryGetObjectsFIFO ( 
    dyn_objects_fifo_t * dof ) [inline], [static]

Returns the pointer to the inner objects FIFO.
6.22 Dynamic Objects Factory

Parameters

| in  | dolp | 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.

6.22.4.28 chFactoryGetPipe()

static pipe_t* chFactoryGetPipe ( 
    dynPipe_t * dpp ) [inline], [static]

Returns the pointer to the inner pipe.

Parameters

| in  | dpp | dynamic pipe object reference |

Returns

The pointer to the pipe.

Function Class:

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

6.22.5 Variable Documentation

6.22.5.1 ch_factory

objects_factory_t ch_factory

Factory object static instance.

Note

It is a global object because it could be accessed through a specific debugger plugin.
Chapter 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:

```
    semaphore_t
      + cnt

    nil_threads_queue
      + cnt

    +sem

    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:

```
    ch_dyn_element
      + refs
      + name

    +next
```

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 next

struct ch_dyn_element* ch_dyn_element::next

Next dynamic object in the list.

7.2.2.2 refs

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.
7.4.1 Detailed Description
Type of a dynamic buffer object.

7.4.2 Field Documentation

7.4.2.1 element

`dyn_element_t ch_dyn_mailbox::element`
List element of the dynamic buffer object.

7.4.2.2 mbx

`mailbox_t ch_dyn_mailbox::mbx`
The mailbox.

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.

7.5.1 Detailed Description

Type of a dynamic buffer object.

7.5.2 Field Documentation

7.5.2.1 element

`dyn_element_t ch_dyn_object::element`

List element of the dynamic buffer object.

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.

7.6.1 Detailed Description

Type of a dynamic buffer object.

7.6.2 Field Documentation
7.6.2.1 element

dyn_element_t ch_dyn_objects_fifo::element

List element of the dynamic buffer object.

7.6.2.2 fifo

objects_fifo_t ch_dyn_objects_fifo::fifo

The objects FIFO.

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.

### 7.7.1 Detailed Description

Type of a dynamic pipe object.
7.7.2 Field Documentation

7.7.2.1 element

dyn_element_t ch_dyn_pipe::element

List element of the dynamic pipe object.

7.7.2.2 pipe

pipe_t ch_dyn_pipe::pipe

The pipe.

7.8 ch_dyn_semaphore Struct Reference

Type of a dynamic semaphore.

#include <chfactory.h>

Collaboration diagram for ch_dyn_semaphore:
7.9 ch_job_descriptor Struct Reference

Type of a job descriptor.

#include <chjobs.h>

Collaboration diagram for ch_job_descriptor:

```
+ ch_job_descriptor
 + jobfunc
 + jobarg
```

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 element

`dyn_element_t ch_dyn_semaphore::element`

List element of the dynamic semaphore.

7.8.2.2 sem

`semaphore_t ch_dyn_semaphore::sem`

The semaphore.
Data Fields

- job_function_t jobfunc
  Job function.
- void* jobarg
  Argument to be passed to the job function.

7.9.1 Detailed Description

Type of a job descriptor.

7.9.2 Field Documentation

7.9.2.1 jobfunc

job_function_t ch_job_descriptor::jobfunc

Job function.

7.9.2.2 jobarg

void* ch_job_descriptor::jobarg

Argument to be passed to the job function.

7.10 ch_jobs_queue Struct Reference

Type of a jobs queue.

#include <chjobs.h>
Collaboration diagram for ch_jobs_queue:

Data Fields

- **guarded_memory_pool_t free**
  
  *Pool of the free jobs.*

- **mailbox_t mbx**
  
  *Mailbox of the sent jobs.*

### 7.10.1 Detailed Description

Type of a jobs queue.

### 7.10.2 Field Documentation
7.10.2.1 free

guarded_memory_pool_t ch_jobs_queue::free

Pool of the free jobs.

7.10.2.2 mbx

mailbox_t ch_jobs_queue::mbx

Mailbox of the sent jobs.

7.11 ch_objects_cache Struct Reference

Structure representing a cache object.

#include <chobjcaches.h>
Collaboration diagram for ch_objects_cache:

```
<table>
<thead>
<tr>
<th>ch_objects_cache</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ hashn</td>
</tr>
<tr>
<td>+ objn</td>
</tr>
<tr>
<td>+ objsz</td>
</tr>
<tr>
<td>+ objvp</td>
</tr>
<tr>
<td>+ readf</td>
</tr>
<tr>
<td>+ writef</td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>nil_threads_queue</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ cnt</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>ch_oc_object</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ obj_group</td>
</tr>
<tr>
<td>+ obj_key</td>
</tr>
<tr>
<td>+ obj_flags</td>
</tr>
<tr>
<td>+ dptr</td>
</tr>
<tr>
<td>+ obj_sem</td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>ch_oc_lru_header</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ lru</td>
</tr>
<tr>
<td>+ lru_prev</td>
</tr>
<tr>
<td>+ lru_next</td>
</tr>
<tr>
<td>+ hash_prev</td>
</tr>
<tr>
<td>+ hash_next</td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>ch_oc_hash_header</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ hashp</td>
</tr>
<tr>
<td>+ hash_prev</td>
</tr>
<tr>
<td>+ hash_next</td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>Data Fields</th>
</tr>
</thead>
<tbody>
<tr>
<td>• ucnt_t hashn</td>
</tr>
<tr>
<td>Number of elements in the hash table.</td>
</tr>
<tr>
<td>• oc_hash_header_t * hashp</td>
</tr>
<tr>
<td>Pointer to the hash table.</td>
</tr>
<tr>
<td>• ucnt_t objn</td>
</tr>
<tr>
<td>Number of elements in the objects table.</td>
</tr>
<tr>
<td>• size_t objsz</td>
</tr>
<tr>
<td>Size of elements in the objects table.</td>
</tr>
</tbody>
</table>
```
7.11 Data Structure Documentation

- void * objvp
  Pointer to the objects table.
- oc_lru_header_t lru
  LRU list header.
- semaphore_t cache_sem
  Semaphore for cache access.
- semaphore_t lru_sem
  Semaphore for LRU access.
- oc_readf_t readf
  Reader functions for cached objects.
- oc_writef_t writef
  Writer functions for cached objects.

7.11.1 Detailed Description

Structure representing a cache object.

7.11.2 Field Documentation

7.11.2.1 hashn

ucnt_t ch_objects_cache::hashn

Number of elements in the hash table.

7.11.2.2 hashp

oc_hash_header_t* ch_objects_cache::hashp

Pointer to the hash table.

7.11.2.3 objn

ucnt_t ch_objects_cache::objn

Number of elements in the objects table.
7.11.2.4  objsz

size_t ch_objects_cache::objsz

Size of elements in the objects table.

7.11.2.5  objvp

void* ch_objects_cache::objvp

Pointer to the objects table.

7.11.2.6  lru

oc_lru_header_t ch_objects_cache::lru

LRU list header.

7.11.2.7  cache_sem

semaphore_t ch_objects_cache::cache_sem

Semaphore for cache access.

7.11.2.8  lru_sem

semaphore_t ch_objects_cache::lru_sem

Semaphore for LRU access.

7.11.2.9  readf

oc_readf_t ch_objects_cache::readf

Reader functions for cached objects.
7.11.2.10 writef

_ac_writef_t ch_objects_cache::writef

Writer functions for cached objects.

7.12 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.

7.12.1 Detailed Description

Type of the factory main object.

7.12.2 Field Documentation

7.12.2.1 mtx

`mutex_t ch_objects_factory::mtx`

Factory access mutex or semaphore.

7.12.2.2 obj_list

`dyn_list_t ch_objects_factory::obj_list`

List of the registered objects.
7.12.2.3 obj_pool

memory_pool_t ch_objects_factory::obj_pool

Pool of the available registered objects.

7.12.2.4 buf_list

dyn_list_t ch_objects_factory::buf_list

List of the allocated buffer objects.

7.12.2.5 sem_list

dyn_list_t ch_objects_factory::sem_list

List of the allocated semaphores.

7.12.2.6 sem_pool

memory_pool_t ch_objects_factory::sem_pool

Pool of the available semaphores.

7.12.2.7 mbx_list

dyn_list_t ch_objects_factory::mbx_list

List of the allocated buffer objects.

7.12.2.8 fifo_list

dyn_list_t ch_objects_factory::fifo_list

List of the allocated "objects FIFO" objects.
7.12.2.9 pipe_list

dyn_list_t ch_objects_factory::pipe_list

List of the allocated pipe objects.

7.13 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.
7.13.1 Detailed Description

Type of an objects FIFO.

7.13.2 Field Documentation

7.13.2.1 free

guarded_memory_pool_t ch_objects_fifo::free

Pool of the free objects.

7.13.2.2 mbx

mailbox_t ch_objects_fifo::mbx

Mailbox of the sent objects.

7.14 ch oc_hash_header Struct Reference

Structure representing an hash table element.

#include <chobjcaches.h>
Collaboration diagram for ch_oc_hash_header:

Data Fields

- `oc_object_t * hash_next
  Next in the collisions list.
- `oc_object_t * hash_prev
  Previous in the collisions list.

7.14.1 Detailed Description

Structure representing an hash table element.

7.14.2 Field Documentation
7.14.2.1 hash_next

oc_object_t* ch_oc_hash_header::hash_next

Next in the collisions list.

7.14.2.2 hash_prev

oc_object_t* ch_oc_hash_header::hash_prev

Previous in the collisions list.

7.15 ch_oc_lru_header Struct Reference

Structure representing an hash table element.

#include <chobjcaches.h>

Collaboration diagram for ch_oc_lru_header:
Data Fields

- `oc_object_t * hash_next
  Next in the collisions list.
- `oc_object_t * hash_prev
  Previous in the collisions list.
- `oc_object_t * lru_next
  Next in the LRU list.
- `oc_object_t * lru_prev
  Previous in the LRU list.

7.15.1 Detailed Description

Structure representing an hash table element.

7.15.2 Field Documentation

7.15.2.1 hash_next

`oc_object_t* ch_oc_lru_header::hash_next

Next in the collisions list.

7.15.2.2 hash_prev

`oc_object_t* ch_oc_lru_header::hash_prev

Previous in the collisions list.

7.15.2.3 lru_next

`oc_object_t* ch_oc_lru_header::lru_next

Next in the LRU list.

7.15.2.4 lru_prev

`oc_object_t* ch_oc_lru_header::lru_prev

Previous in the LRU list.
7.16  ch_oc_object Struct Reference

Structure representing a cached object.

#include <chobjcaches.h>

Collaboration diagram for ch_oc_object:

Data Fields

- oc_object_t * hash_next
  Next in the collisions list.
- oc_object_t * hash_prev
  Previous in the collisions list.
- oc_object_t * lru_next
  Next in the LRU list.
- oc_object_t * lru_prev
  Previous in the LRU list.
- uint32_t obj_group
  Object group.
- uint32_t obj_key
  Object key.
- semaphore_t obj_sem
  Semaphore for object access.
- oc_flags_t obj_flags
  Object flags.
- void * dptr
  User pointer.
7.16 ch_oc_object Struct Reference

7.16.1 Detailed Description

Structure representing a cached object.

7.16.2 Field Documentation

7.16.2.1 hash_next

\*oc_object_t ch_oc_object::hash_next

Next in the collisions list.

7.16.2.2 hash_prev

\*oc_object_t ch_oc_object::hash_prev

Previous in the collisions list.

7.16.2.3 lru_next

\*oc_object_t ch_oc_object::lru_next

Next in the LRU list.

7.16.2.4 lru_prev

\*oc_object_t ch_oc_object::lru_prev

Previous in the LRU list.

7.16.2.5 obj_group

\*uint32_t ch_oc_object::obj_group

Object group.
7.16.2.6  **obj_key**

```c
uint32_t ch_oc_object::obj_key
```

Object key.

7.16.2.7  **obj_sem**

```c
semaphore_t ch_oc_object::obj_sem
```

Semaphore for object access.

7.16.2.8  **obj_flags**

```c
oc_flags_t ch_oc_object::obj_flags
```

Object flags.

7.16.2.9  **dptr**

```c
void* ch_oc_object::dptr
```

User pointer.

**Note**

This pointer can be used to refer to external buffers, `chCacheObjectInit()` initializes it to `NULL`. 
7.17  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.

7.17.1 Detailed Description

Type of a registered object.

7.17.2 Field Documentation

7.17.2.1 element

dyn_element_t  ch_registered_static_object::element

List element of the registered object.
7.17.2.2 objp

```c
void* ch_registered_static_object::objp
```

Pointer to the object.

**Note**

The type of the object is not stored in anyway.

---

### 7.18 event_listener Struct Reference

Event Listener structure.

```c
#include <chevt.h>
```

Collaboration diagram for event_listener:
Data Fields

- `event_listener_t * next`
  Next Event Listener registered on the event source.

- `thread_t * listener`
  Thread interested in the event source.

- `eventmask_t events`
  Events to be set in the listening thread.

- `eventflags_t flags`
  Flags added to the listener by the event source.

- `eventflags_t wflags`
  Flags that this listener interested in.

7.18.1 Detailed Description

Event Listener structure.

7.18.2 Field Documentation

7.18.2.1 next

```
event_listener_t* event_listener::next
```

Next Event Listener registered on the event source.

7.18.2.2 listener

```
thread_t* event_listener::listener
```

Thread interested in the event source.

7.18.2.3 events

```
eventmask_t event_listener::events
```

Events to be set in the listening thread.
7.18.2.4 flags

```c
eventflags_t event_listener::flags
```

Flags added to the listener by the event source.

7.18.2.5 wflags

```c
eventflags_t event_listener::wflags
```

Flags that this listener interested in.

7.19 event_source Struct Reference

Event Source structure.

```c
#include <chevt.h>
```
Collaboration diagram for event_source:

Data Fields

- `event_listener_t ∗ next`
  
  *First Event Listener registered on the Event Source.*

7.19.1 Detailed Description

Event Source structure.

7.19.2 Field Documentation
7.19.2.1 next

`event_listener_t* event_source::next`

First Event Listener registered on the Event Source.

7.20 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.
```
7.20.1 Detailed Description

Guarded memory pool descriptor.

7.20.2 Field Documentation

7.20.2.1 sem

semaphore_t guarded_memory_pool_t::sem

Counter semaphore guarding the memory pool.

7.20.2.2 pool

memory_pool_t guarded_memory_pool_t::pool

The memory pool itself.

7.21 heap_header Union Reference

Memory heap block header.

#include <chmemheaps.h>

Collaboration diagram for heap_header:

![Collaboration diagram for heap_header]
7.21.1 Detailed Description

Memory heap block header.

7.21.2 Field Documentation

7.21.2.1 next

heap_header_t* heap_header::next

Next block in free list.

7.21.2.2 pages

size_t heap_header::pages

Size of the area in pages.

7.21.2.3 heap

memory_heap_t* heap_header::heap

Block owner heap.

7.21.2.4 size

size_t heap_header::size

Size of the area in bytes.
7.22 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

ChibiOS/NIL
Queued writers.

- threads_queue_t qr

Queued readers.

7.22.1 Detailed Description

Structure representing a mailbox object.

7.22.2 Field Documentation

7.22.2.1 buffer

msg_t* mailbox_t::buffer

Pointer to the mailbox buffer.

7.22.2.2 top

msg_t* mailbox_t::top

Pointer to the location after the buffer.

7.22.2.3 wrptr

msg_t* mailbox_t::wrptr

Write pointer.

7.22.2.4 rdptr

msg_t* mailbox_t::rdptr

Read pointer.
7.22.2.5  cnt

size_t mailbox_t::cnt

Messages in queue.

7.22.2.6  reset

bool mailbox_t::reset

True in reset state.

7.22.2.7  qw

threads_queue_t mailbox_t::qw

Queued writers.

7.22.2.8  qr

threads_queue_t mailbox_t::qr

Queued readers.

7.23  memcore_t Struct Reference

Type of memory core object.

#include <chmemcore.h>

Collaboration diagram for memcore_t:

```
+----+----+
| memcore_t |
| + basemem |
| + topmem  |
```
Data Structure Documentation

Data Fields

- `uint8_t * basemem`
  
  Next free address.
- `uint8_t * topmem`
  
  Final address.

7.23.1 Detailed Description

Type of memory core object.

7.23.2 Field Documentation

7.23.2.1 basemem

`uint8_t* memcore_t::basemem`

Next free address.

7.23.2.2 topmem

`uint8_t* memcore_t::topmem`

Final address.

7.24 memory_heap Struct Reference

Structure describing a memory heap.

#include <chmemheaps.h>
Collaboration diagram for `memory_heap`:

![Collaboration Diagram for memory_heap](image)

**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.

**7.24.1 Detailed Description**

Structure describing a memory heap.

**7.24.2 Field Documentation**

7.24.2.1 provider

`memgetfunc2_t memory_heap::provider`

Memory blocks provider for this heap.
7.24.2.2 header

`heap_header_t memory_heap::header`

Free blocks list header.

7.24.2.3 mtx

`mutex_t memory_heap::mtx`

Heap access mutex.

7.25 `memory_pool_t` Struct Reference

Memory pool descriptor.

```c
#include <chmempools.h>
```

Collaboration diagram for `memory_pool_t`:

![Collaboration Diagram](image-url)
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.

7.25.1 Detailed Description

Memory pool descriptor.

7.25.2 Field Documentation

7.25.2.1 next

struct pool_header* memory_pool_t::next

Pointer to the header.

7.25.2.2 object_size

size_t memory_pool_t::object_size

Memory pool objects size.

7.25.2.3 align

unsigned memory_pool_t::align

Required alignment.
7.25.2.4 provider

`memgetfunc_t memory_pool_t::provider`

Memory blocks provider for this pool.

7.26 nil_os_instance Struct Reference

System data structure.

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

Collaboration diagram for nil_os_instance:

![Collaboration diagram for nil_os_instance](image)

Data Fields

- `thread_t * current`
  
  Pointer to the running thread.
- `thread_t * next`
7.26.1 Detailed Description

System data structure.

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

7.26.2 Field Documentation

7.26.2.1 current

thread_t* nil_os_instance::current

Pointer to the running thread.

7.26.2.2 next

thread_t* nil_os_instance::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.
7.26.2.3 systime

volatile systime_t nil_os_instance::systime

System time.

7.26.2.4 lasttime

systime_t nil_os_instance::lasttime

System time of the last tick event.

7.26.2.5 nexttime

systime_t nil_os_instance::nexttime

Time of the next scheduled tick event.

7.26.2.6 isr_cnt

cnt_t nil_os_instance::isr_cnt

ISR nesting level.

7.26.2.7 lock_cnt

cnt_t nil_os_instance::lock_cnt

Lock nesting level.

7.26.2.8 dbg_panic_msg

const char* volatile nil_os_instance::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.
7.26.2.9 threads

thread_t nil_os_instance::threads[CH_CFG_MAX_THREADS+1]

Thread structures for all the defined threads.

7.27 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.

- msg_t sntmsg
  Sent message.

- stkalign_t *wabase
  Thread stack boundary.

- msg_t msg
  Wake-up/exit message.

- void *p
  Generic pointer.

- os_instance_t *nsp
  Pointer to nil base struct.

- thread_reference_t *trp
  Pointer to thread reference.

- threads_queue_t *tqp
  Pointer to thread queue.

- thread_t *tp
  Pointer to thread.

- semaphore_t *semp
  Pointer to semaphore.

- eventmask_t ewmask
  Enabled events mask.

7.27.1 Detailed Description

Structure representing a thread.

7.27.2 Field Documentation
7.27.1 ctx

`struct port_context nil_thread::ctx`

Processor context.

7.27.2 state

`tstate_t nil_thread::state`

Thread state.

7.27.3 msg

`msg_t nil_thread::msg`

Wake-up/exit message.

7.27.4 p

`void* nil_thread::p`

Generic pointer.

7.27.5 nsp

`os_instance_t* nil_thread::nsp`

Pointer to nil base struct.

7.27.6 trp

`thread_reference_t* nil_thread::trp`

Pointer to thread reference.
7.27.2.7  tqp

threads_queue_t* nil_thread::tqp

Pointer to thread queue.

7.27.2.8  tp

thread_t* nil_thread::tp

Pointer to thread.

7.27.2.9  semp

semaphore_t* nil_thread::semp

Pointer to semaphore.

7.27.2.10  ewmask

eventmask_t nil_thread::ewmask

Enabled events mask.

7.27.2.11  timeout

volatile sysinterval_t nil_thread::timeout

Timeout counter, zero if disabled.

7.27.2.12  epmask

eventmask_t nil_thread::epmask

Pending events mask.
### 7.28 nil_thread_descriptor Struct Reference

Structure representing a thread descriptor.

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

Collaboration diagram for nil_thread_descriptor:

```
nil_thread_descriptor
+ name
+ wbase
+ wend
+ prio
+ funcp
+ arg
```

#### Data Fields

- **const char * name**
  
  Thread name, for debugging.

- **stkalign_t * wbase**
  
  Thread working area base.

- **stkalign_t * wend**
  
  Thread working area end.
• *tprio_t prio
  Thread priority slot.

• *tfunc_t funcp
  Thread function.

• *void arg
  Thread function argument.

7.28.1 Detailed Description

Structure representing a thread descriptor.

7.28.2 Field Documentation

7.28.2.1 name

const char* nil_thread_descriptor::name

Thread name, for debugging.

7.28.2.2 wbase

stkalign_t* nil_thread_descriptor::wbase

Thread working area base.

7.28.2.3 wend

stkalign_t* nil_thread_descriptor::wend

Thread working area end.
7.28.2.4  prio

tprio_t nil_thread_descriptor::prio

Thread priority slot.

7.28.2.5  funcp

tfunc_t nil_thread_descriptor::funcp

Thread function.

7.28.2.6  arg

void* nil_thread_descriptor::arg

Thread function argument.

7.29  nil_threads_queue Struct Reference

Structure representing a queue of threads.

#include <ch.h>

Inheritance diagram for nil_threads_queue:
Data Fields

- volatile cnt_t cnt

  Threads Queue counter.

7.29.1 Detailed Description

Structure representing a queue of threads.

7.29.2 Field Documentation

7.29.2.1 cnt

volatile cnt_t nil_threads_queue::cnt

Threads Queue counter.

7.30 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.
7.30.1 Detailed Description

Structure representing a pipe object.

7.30.2 Field Documentation

7.30.2.1 buffer

```c
uint8_t* pipe_t::buffer
```

Pointer to the pipe buffer.

7.30.2.2 top

```c
uint8_t* pipe_t::top
```

Pointer to the location after the buffer.
7.30.2.3 wrptr

```
uint8_t* pipe_t::wrptr
```

Write pointer.

7.30.2.4 rdptr

```
uint8_t* pipe_t::rdptr
```

Read pointer.

7.30.2.5 cnt

```
size_t pipe_t::cnt
```

Bytes in the pipe.

7.30.2.6 reset

```
bool pipe_t::reset
```

True if in reset state.

7.30.2.7 wtr

```
thread_reference_t pipe_t::wtr
```

Waiting writer.

7.30.2.8 rtr

```
thread_reference_t pipe_t::rtr
```

Waiting reader.
7.30.2.9  cmtx

mutex_t pipe_t::cmtx

Common access mutex.

7.30.2.10  wmtx

mutex_t pipe_t::wmtx

Write access mutex.

7.30.2.11  rmtx

mutex_t pipe_t::rmtx

Read access mutex.

7.31  pool_header Struct Reference

Memory pool free object header.

#include <chmempools.h>

Collaboration diagram for pool_header:

![Collaboration diagram for pool_header]

Data Fields

- struct pool_header * next

  Pointer to the next pool header in the list.
7.31 pool_header Struct Reference

7.31.1 Detailed Description

Memory pool free object header.

7.31.2 Field Documentation

7.31.2.1 next

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

- `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.
- `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 chSchDoPreemption (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.

- **bool chTimeIsInRangeX (systime_t time, systime_t start, systime_t end)**
  
  Checks if the specified time is within the specified time range.

- **thread_t * chThdCreateI (const thread_descriptor_t *tdp)**
  
  Creates a new thread into a static memory area.

- **thread_t * chThdCreate (const thread_descriptor_t *tdp)**
  
  Creates a new thread into a static memory area.

- **void chThdExit (msg_t msg)**
  
  Terminates the current thread.

- **msg_t chThdWait (thread_t *tp)**
  
  Blocks the execution of the invoking thread until the specified thread terminates then the exit code is returned.

- **msg_t 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 (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.
8.2 ch.h File Reference

- 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.

Variables

- os_instance_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 "chsem.h"
#include "chevt.h"
#include "chmsg.h"
#include "chlib.h"

Data Structures

- struct nil_threads_queue
  Structure representing a queue of threads.
- struct nil_thread_descriptor
  Structure representing a thread descriptor.
- struct nil_thread
  Structure representing a thread.
- struct nil_os_instance
  System data structure.
Macros

- `#define __CHIBIOS_NIL__`
  ChibiOS/NIL identification macro.
- `#define CH_KERNEL_STABLE 0`
  Stable release flag.
- `#define THD_IDLE_BASE (&__main_thread_stack_base__)`
- `#define __CH_STRINGIFY(a) #a`
  Utility to make the parameter a quoted string.
- `#define THD_WORKING_AREA_END(wa) ((wa) + ((sizeof wa) / sizeof (stkalign_t)))`
  Returns the top address of a working area.

ChibiOS/NIL version identification

- `#define CH_KERNEL_VERSION "4.1.0"`
  Kernel version string.
- `#define CH_KERNEL_MAJOR 4`
  Kernel version major number.
- `#define CH_KERNEL_MINOR 1`
  Kernel version minor number.
- `#define CH_KERNEL_PATCH 0`
  Kernel version patch number.

Constants for configuration options

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

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_WTSTART (tstate_t) 0
  Thread not yet started or terminated.

• #define NIL_STATE_READY (tstate_t) 1
  Thread ready or executing.

• #define NIL_STATE_SLEEPING (tstate_t) 2
  Thread sleeping.

• #define NIL_STATE_SUSPENDED (tstate_t) 3
  Thread suspended.

• #define NIL_STATE_WTEXIT (tstate_t) 4
  Waiting a thread.

• #define NIL_STATE_WTQUEUE (tstate_t) 5
  On queue or semaph.

• #define NIL_STATE_WTOREVT (tstate_t) 6
  Waiting for events.

• #define NIL_STATE_WTANDEVT (tstate_t) 7
  Waiting for events.

• #define NIL_STATE_SNDMSGQ (tstate_t) 8
  Sending a message, in queue.

• #define NIL_STATE_WTMSG (tstate_t) 10
  Waiting for a message.

• #define NIL_STATE_FINAL (tstate_t) 11
  Thread terminated.

• #define NIL_THD_IS_WTSTART(tp) ((tp)->state == NIL_STATE_WTSTART)
• #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_SUSPENDED(tp) ((tp)->state == NIL_STATE_SUSPENDED)
• #define NIL_THD_IS_WTEXIT(tp) ((tp)->state == NIL_STATE_WTEXIT)
• #define NIL_THD_IS_WTQUEUE(tp) ((tp)->state == NIL_STATE_WTQUEUE)
• #define NIL_THD_IS_WTOREVT(tp) ((tp)->state == NIL_STATE_WTOREVT)
• #define NIL_THD_IS_WTANDEVT(tp) ((tp)->state == NIL_STATE_WTANDEVT)
• #define NIL_THD_IS_SNDMSGQ(tp) ((tp)->state == NIL_STATE_SNDMSGQ)
• #define NIL_THD_IS_WTMSG(tp) ((tp)->state == NIL_STATE_WTMSG)
• #define NIL_THD_IS_FINAL(tp) ((tp)->state == NIL_STATE_FINAL)
• #define CH_STATE_NAMES

RT options not existing in NIL

• #define CH_CFG_USE_REGISTRY FALSE

Threads tables definition macros

• #define THD_TABLE_BEGIN const thread_descriptor_t nil_thd_configs[] = {
  Start of user threads table.
• #define THD_TABLE_THREAD(_prio, _name, _wap, _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.

**ISR 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.

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 chSchGoSleepS(newstate) chSchGoSleepTimeoutS(newstate, TIME_INFINITE)  
  Puts the current thread to sleep into the specified state.
- \#define chSchWakeupS(ntp, msg)  
  Wakes up a thread.
- \#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 chThdGetPriorityX(void) (tprio_t)(nil.current - &nil.threads[0])  
  Returns the current thread priority.
- \#define chThdResumeS(trp, msg)  
  Wakes up a thread waiting on a thread reference object.
- \#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 \textit{true} if the specified queue is empty.
- \#define chVTGetSystemTimeX() (nil.systime)  
  Current system time.
- \#define chVTTimeElapsedSinceX(start) chTimeDiffX((start), chVTGetSystemTimeX())  
  Returns the elapsed time since the specified start time.
- \#define chVTIsSystemTimeWithinX(start, end) chTimeIsInRangeX(chVTGetSystemTimeX(), start, end)  
  Checks if the current system time is within the specified time window.
- \#define chTimeAddX(systime, interval) ((systime_t)(systime) + (systime_t)(interval))  
  Adds an interval to a system time returning a system time.
• `#define chTimeDiffX(start, end) ((sysinterval_t)((systime_t)((systime_t)(end) - (systime_t)(start))))`  
  Subtracts two system times returning an interval.
• `#define chDbgCheck(c)`  
  Function parameters check.
• `#define chDbgAssert(c, r)`  
  Condition assertion.

**Typedefs**

- `typedef uint32_t systime_t`  
  Type of system time.
- `typedef uint32_t sysinterval_t`  
  Type of time interval.
- `typedef uint64_t time_conv_t`  
  Type of time conversion variable.
- `typedef struct nil_os_instance os_instance_t`  
  Type of a structure representing the system.
- `typedef void (∗ tfunc_t) (void ∗ p)`  
  Thread function.
- `typedef struct nil_thread_descriptor thread_descriptor_t`  
  Type of a thread descriptor.
- `typedef struct nil_thread thread_t`  
  Type of a structure representing a thread.
- `typedef thread_t ∗ thread_reference_t`  
  Type of a thread reference.
- `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.

**Kernel types**

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

**Functions**

- `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.
- `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 chSysInit (void)`  
  Initializes the kernel.
8.2 ch.h File Reference

- void chSysHalt (const char *reason)
  Halts the system.
- void chSysTimerHandlerI (void)
  Time management handler.
- void chSysUnconditionalLock (void)
  Unconditionally enters the kernel lock state.
- void chSysUnconditionalUnlock (void)
  Unconditionally leaves the kernel lock state.
- syssts_t chSysGetStatusAndLockX (void)
  Returns the execution status and enters a critical zone.
- bool chSysIsCounterWithinX (rtcnt_t cnt, 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 chSchDoPreemption (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.
- bool chTimelsInRangeX (systime_t time, systime_t start, systime_t end)
  Checks if the specified time is within the specified time range.
- thread_t * chThdCreateI (const thread_descriptor_t *tdp)
  Creates a new thread into a static memory area.
- thread_t * chThdCreate (const thread_descriptor_t *tdp)
  Creates a new thread into a static memory area.
- void chThdExit (msg_t msg)
  Terminates the current thread.
- 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 (systime_t abstime)
  Suspends the invoking thread until the system time arrives to the specified value.
- msg_t chThdEnqueueTimeoutS (threads_queue_t *tqp, sysinterval_t timeout)
  Enqueues the caller thread on a threads queue object.
- void chThdDoDequeueNextI (threads_queue_t *tqp, msg_t msg)
  Dequeues and wakes up one thread from the threads queue object.
- void chThdDoDequeueNextI (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.
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 BSEMAPHOREDECL(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 a 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 chBSemGetStateI (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_MAX_THREADS 4`
  
  Maximum number of user threads in the application.
- `#define CH_CFG_AUTOSTART_THREADS TRUE`
  
  Auto starts threads when `chSysInit()` is invoked.

**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_WAITEXIT TRUE`
Threads synchronization APIs.
• #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_MESSAGES TRUE
  Synchronous Messages APIs.

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

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

Debug options
• #define CH_DBG_STATISTICS FALSE
  Debug option, kernel statistics.
• #define CH_DBG_SYSTEM_STATE_CHECK TRUE
Debug option, system state check.
- #define CH_DBG_ENABLE_CHECKS TRUE
Debug option, parameters checks.
- #define CH_DBG_ENABLE_ASSERTS TRUE
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_THREAD_EXIT_HOOK(tp) {}
  Threads finalization 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 chdelegates.c File Reference

Delegate threads code.
#include "ch.h"

Functions
- msg_t __ch_delegate_fn0 (va_list *argsp)
  Veneer for functions with no parameters.
- msg_t __ch_delegate_fn1 (va_list *argsp)
  Veneer for functions with one parameter.
- msg_t __ch_delegate_fn2 (va_list *argsp)
  Veneer for functions with two parameters.
- msg_t __ch_delegate_fn3 (va_list *argsp)
  Veneer for functions with three parameters.
- msg_t __ch_delegate_fn4 (va_list *argsp)
  Veneer for functions with four parameters.
- msg_t chDelegateCallVeneer (thread_t *tp, delegate_veneer_t veneer,...)
  Triggers a function call on a delegate thread.
- void chDelegateDispatch (void)
  Call messages dispatching.
- msg_t chDelegateDispatchTimeout (sysinterval_t timeout)
  Call messages dispatching with timeout.
8.5.1 Detailed Description

Delegate threads code.

Delegate threads.

Operation mode

A delegate thread is a thread performing function calls triggered by other threads. This functionality is especially useful when encapsulating a library not designed for threading into a delegate thread. Other threads have access to the library without having to worry about mutual exclusion.

Precondition

In order to use the pipes APIs the CH_CFG_USE_DELEGATES option must be enabled in chconf.h.

Note

Compatible with RT and NIL.

8.6 chdelegates.h File Reference

Delegate threads macros and structures.

#include <stdarg.h>

Typedefs

- typedef msg_t(* delegate_veneer_t) (va_list *argsp)
  Type of a delegate veneer function.
- typedef msg_t(* delegate_fn0_t) (void)
  Type of a delegate function with no parameters.
- typedef msg_t(* delegate_fn1_t) (msg_t p1)
  Type of a delegate function with one parameter.
- typedef msg_t(* delegate_fn2_t) (msg_t p1, msg_t p2)
  Type of a delegate function with two parameters.
- typedef msg_t(* delegate_fn3_t) (msg_t p1, msg_t p2, msg_t p3)
  Type of a delegate function with three parameters.
- typedef msg_t(* delegate_fn4_t) (msg_t p1, msg_t p2, msg_t p3, msg_t p4)
  Type of a delegate function with four parameters.
Functions

- `msg_t ch_delegate_fn0 (va_list *argsp)`
  Veneer for functions with no parameters.
- `msg_t ch_delegate_fn1 (va_list *argsp)`
  Veneer for functions with one parameter.
- `msg_t ch_delegate_fn2 (va_list *argsp)`
  Veneer for functions with two parameters.
- `msg_t ch_delegate_fn3 (va_list *argsp)`
  Veneer for functions with three parameters.
- `msg_t ch_delegate_fn4 (va_list *argsp)`
  Veneer for functions with four parameters.
- `void chDelegateDispatch (void)`
  Call messages dispatching.
- `msg_t chDelegateDispatchTimeout (sysinterval_t timeout)`
  Call messages dispatching with timeout.
- `msg_t chDelegateCallVeneer (thread_t *tp, delegate_veneer_t veneer,...)`
  Triggers a function call on a delegate thread.
- `static msg_t chDelegateCallDirect0 (thread_t *tp, delegate_fn0_t func)`
  Direct call to a function with no parameters.
- `static msg_t chDelegateCallDirect1 (thread_t *tp, delegate_fn1_t func, msg_t p1)`
  Direct call to a function with one parameter.
- `static msg_t chDelegateCallDirect2 (thread_t *tp, delegate_fn2_t func, msg_t p1, msg_t p2)`
  Direct call to a function with two parameters.
- `static msg_t chDelegateCallDirect3 (thread_t *tp, delegate_fn3_t func, msg_t p1, msg_t p2, msg_t p3)`
  Direct call to a function with three parameters.
- `static msg_t chDelegateCallDirect4 (thread_t *tp, delegate_fn4_t func, msg_t p1, msg_t p2, msg_t p3, msg_t p4)`
  Direct call to a function with four parameters.

8.6.1 Detailed Description

Delegate threads macros and structures.

8.7 chevt.c File Reference

Nil RTOS events source file.

#include "ch.h"
Functions

- void chEvtRegisterMaskWithFlags (event_source_t *esp, event_listener_t *elp, eventmask_t events, eventflags_t wflags)
  Registers an Event Listener on an Event Source.
- void chEvtUnregister (event_source_t *esp, event_listener_t *elp)
  Unregisters an Event Listener from its Event Source.
- eventmask_t chEvtGetAndClearEventsI (eventmask_t events)
  Clears the pending events specified in the events mask.
- eventmask_t chEvtGetAndClearEvents (eventmask_t events)
  Clears the pending events specified in the events mask.
- eventmask_t chEvtAddEvents (eventmask_t events)
  Adds (OR) a set of events to the current thread, this is much faster than using chEvtBroadcast() or chEvtSignal().
- void chEvtBroadcastFlagsI (event_source_t *esp, eventflags_t flags)
  Signals all the Event Listeners registered on the specified Event Source.
- eventflags_t chEvtGetAndClearFlags (event_listener_t *elp)
  Returns the flags associated to an event_listener_t.
- void chEvtSignal (thread_t *tp, eventmask_t events)
  Adds a set of event flags directly to the specified thread_t.
- void chEvtSignall (thread_t *tp, eventmask_t events)
  Adds a set of event flags directly to the specified thread_t.
- void chEvtBroadcastFlags (event_source_t *esp, eventflags_t flags)
  Signals all the Event Listeners registered on the specified Event Source.
- eventflags_t chEvtGetAndClearFlagsI (event_listener_t *elp)
  Returns the unmasked flags associated to an event_listener_t.
- void chEvtDispatch (const evhandler_t *handlers, eventmask_t events)
  Invokes the event handlers associated to an event flags mask.
- eventmask_t chEvtWaitOneTimeout (eventmask_t events, sysinterval_t timeout)
  Waits for exactly one of the specified events.
- eventmask_t chEvtWaitAnyTimeout (eventmask_t mask, sysinterval_t timeout)
  Waits for any of the specified events.
- eventmask_t chEvtWaitAllTimeout (eventmask_t mask, sysinterval_t timeout)
  Waits for all the specified events.

8.7.1 Detailed Description

Nil RTOS events source file.

8.8 chevt.h File Reference

Nil RTOS events header file.

Data Structures

- struct event_listener
  Event Listener structure.
- struct event_source
  Event Source structure.
Macros

- `#define ALL_EVENTS ((eventmask_t)-1)`
  All events allowed mask.
- `#define EVENT_MASK(eid) ((eventmask_t)1 << (eventmask_t)(eid))`
  Returns an event mask from an event identifier.
- `#define _EVENTSOURCE_DATA(name) {(event_listener_t *)(&name)}`
  Data part of a static event source initializer.
- `#define EVENTSOURCE_DECL(name) event_source_t name = _EVENTSOURCE_DATA(name)`
  Static event source initializer.

Macro Functions

- `#define chEvtObjectInit(esp)`
  Initializes an Event Source.
- `#define chEvtRegisterMask(esp, elp, events) chEvtRegisterMaskWithFlags(esp, elp, events, (eventflags_t)-1)`
  Registers an Event Listener on an Event Source.
- `#define chEvtRegister(esp, elp, event) chEvtRegisterMask(esp, elp, EVENT_MASK(event))`
  Registers an Event Listener on an Event Source.
- `#define chEvtIsListeningI(esp) (bool)((esp) != (event_source_t*)(esp)->next)`
  Verifies if there is at least one event_listener_t registered.
- `#define chEvtBroadcastI(esp) chEvtBroadcastFlagsI(esp, (eventflags_t)0)`
  Signals all the Event Listeners registered on the specified Event Source.
- `#define chEvtBroadcast(esp) chEvtBroadcastFlags(esp, (eventflags_t)0)`
  Signals all the Event Listeners registered on the specified Event Source.
- `#define chEvtAddEventsI(events) (nil.current->epmask |= events)`
  Adds (OR) a set of events to the current thread, this is much faster than using chEvtBroadcast() or chEvtSignal().
- `#define chEvtGetEventsX(void) (nil.current->epmask)`
  Returns the events mask.
- `#define chEvtWaitOne(events) chEvtWaitOneTimeout(events, TIME_INFINITE)`
  Waits for exactly one of the specified events.
- `#define chEvtWaitAny(events) chEvtWaitAnyTimeout(events, TIME_INFINITE)`
  Waits for any of the specified events.
- `#define chEvtWaitAll(events) chEvtWaitAllTimeout(events, TIME_INFINITE)`
  Waits for all the specified events.

Typedefs

- `typedef struct event_source event_source_t`
  Event Source structure.
- `typedef void (*)(eventid_t id) (eventid_t id)`
  Event Handler callback function.

Functions

- `void chEvtRegisterMaskWithFlags (event_source_t *esp, event_listener_t *elp, eventmask_t events, eventflags_t wflags)`
  Registers an Event Listener on an Event Source.
- `void chEvtUnregister (event_source_t *esp, event_listener_t *elp)`
  Unregisters an Event Listener from its Event Source.
- `eventmask_t chEvtGetAndClearEventsI (eventmask_t events)`
Clears the pending events specified in the events mask.

- `eventmask_t chEvtGetAndClearEvents (eventmask_t events)`
  Clears the pending events specified in the events mask.

- `eventmask_t chEvtAddEvents (eventmask_t events)`
  Adds (OR) a set of events to the current thread, this is much faster than using `chEvtBroadcast()` or `chEvtSignal()`.

- `eventflags_t chEvtGetAndClearFlags (event_listener_t *elp)`
  Returns the flags associated to an `event_listener_t`.

- `eventflags_t chEvtGetAndClearFlagsI (event_listener_t *elp)`
  Returns the unmasked flags associated to an `event_listener_t`.

- `void chEvtSignal (thread_t *tp, eventmask_t events)`
  Adds a set of event flags directly to the specified `thread_t`.

- `void chEvtSignalI (thread_t *tp, eventmask_t events)`
  Adds a set of event flags directly to the specified `thread_t`.

- `void chEvtBroadcastFlags (event_source_t *esp, eventflags_t flags)`
  Signals all the Event Listeners registered on the specified Event Source.

- `void chEvtBroadcastFlagsI (event_source_t *esp, eventflags_t flags)`
  Signals all the Event Listeners registered on the specified Event Source.

- `void chEvtDispatch (const evhandler_t *handlers, eventmask_t events)`
  Invokes the event handlers associated to an event flags mask.

- `eventmask_t chEvtWaitOneTimeout (eventmask_t events, sysinterval_t timeout)`
  Waits for exactly one of the specified events.

- `eventmask_t chEvtWaitAnyTimeout (eventmask_t mask, sysinterval_t timeout)`
  Waits for any of the specified events.

- `eventmask_t chEvtWaitAllTimeout (eventmask_t mask, sysinterval_t timeout)`
  Waits for all the specified events.

### 8.8.1 Detailed Description

Nil RTOS events header file.

### 8.9 chfactory.c File Reference

ChibiOS objects factory and registry code.

```c
#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.
8.10 chfactory.h File Reference

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.

8.9.1 Detailed Description

ChibiOS objects factory and registry code.

8.10 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_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.
8.10 chfactory.h File Reference

**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.
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` * chFactoryFindObjectFIFO (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` * chFactoryFindObjectPipe (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 * chFactoryFindObjectFIFO (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.

8.10.1 Detailed Description

ChibiOS objects factory structures and macros.

8.11 chjobs.h File Reference

Jobs Queues structures and macros.

Data Structures

• struct ch_jobs_queue
  Type of a jobs queue.

• struct ch_job_descriptor
  Type of a job descriptor.
 Macros

- `#define MSG_JOB_NULL ((msg_t)-2)`
  
  Dispatcher return code in case of a JOB_NULL has been received.

 Typedefs

- `typedef struct ch_jobs_queue jobs_queue_t`
  
  Type of a jobs queue.

- `typedef void(* job_function_t)(void *arg)`
  
  Type of a job function.

- `typedef struct ch_job_descriptor job_descriptor_t`
  
  Type of a job descriptor.

 Functions

- `static void chJobObjectInit (jobs_queue_t *jqp, size_t jobsn, job_descriptor_t *jobsbuf, msg_t *msgbuf)`
  
  Initializes a jobs queue object.

- `static job_descriptor_t * chJobGet (jobs_queue_t *jqp)`
  
  Allocates a free job object.

- `static job_descriptor_t * chJobGetI (jobs_queue_t *jqp)`
  
  Allocates a free job object.

- `static job_descriptor_t * chJobGetTimeoutS (jobs_queue_t *jqp, sysinterval_t timeout)`
  
  Allocates a free job object.

- `static job_descriptor_t * chJobGetTimeout (jobs_queue_t *jqp, sysinterval_t timeout)`
  
  Allocates a free job object.

- `static void chJobPostI (jobs_queue_t *jqp, job_descriptor_t *jp)`
  
  Posts a job object.

- `static void chJobPostS (jobs_queue_t *jqp, job_descriptor_t *jp)`
  
  Posts a job object.

- `static void chJobPost (jobs_queue_t *jqp, job_descriptor_t *jp)`
  
  Posts a job object.

- `static void chJobPostAheadI (jobs_queue_t *jqp, job_descriptor_t *jp)`
  
  Posts an high priority job object.

- `static void chJobPostAheadS (jobs_queue_t *jqp, job_descriptor_t *jp)`
  
  Posts an high priority job object.

- `static void chJobPostAhead (jobs_queue_t *jqp, job_descriptor_t *jp)`
  
  Posts an high priority job object.

- `static msg_t chJobDispatch (jobs_queue_t *jqp)`
  
  Waits for a job then executes it.

- `static msg_t chJobDispatchTimeout (jobs_queue_t *jqp, sysinterval_t timeout)`
  
  Waits for a job then executes it.

8.11 Detailed Description

Jobs Queues structures and macros.

This module implements queues of generic jobs to be delegated asynchronously to a pool of dedicated threads. Operations defined for Jobs Queues

- **Get**: An job object is taken from the pool of the available jobs.

- **Post**: A job is posted to the queue, it will be returned to the pool after execution.
8.12 chlib.h File Reference

ChibiOS/LIB main include file.

```c
#include "chbsem.h"
#include "chmboxes.h"
#include "chmemcore.h"
#include "chmemheaps.h"
#include "chmempools.h"
#include "chobjfifos.h"
#include "chpipes.h"
#include "chobjcaches.h"
#include "chdelegates.h"
#include "chjobs.h"
#include "chfactory.h"
```

Macros

- `#define __CHIBIOS_OSLIB__`
  ChibiOS/LIB identification macro.
- `#define CH_OSLIB_STABLE 0`
  Stable release flag.

ChibiOS/LIB version identification

- `#define CH_OSLIB_VERSION "1.3.0"`
  OS Library version string.
- `#define CH_OSLIB_MAJOR 1`
  OS Library version major number.
- `#define CH_OSLIB_MINOR 3`
  OS Library version minor number.
- `#define CH_OSLIB_PATCH 0`
  OS Library version patch number.

Functions

- `static void __oslib_init (void)`
  Initialization of all library modules.

8.12.1 Detailed Description

ChibiOS/LIB main include file.

This header includes all the required library headers. This file is meant do be included by `ch.h` not directly by user.
# 8.13 chmboxes.c File Reference

Mailboxes code.

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

## Functions

- **void chMBOBJECTInit (mailbox_t *mbp, msg_t *buf, size_t n)**
  
  Initializes a `mailbox_t` object.

- **void chMBReset (mailbox_t *mbp)**
  
  Resets a `mailbox_t` object.

- **void chMBResetI (mailbox_t *mbp)**
  
  Resets a `mailbox_t` object.

- **msg_t chMBPostTimeout (mailbox_t *mbp, msg_t msg, sysinterval_t timeout)**
  
  Posts a message into a mailbox.

- **msg_t chMBPostTimeoutS (mailbox_t *mbp, msg_t msg, sysinterval_t timeout)**
  
  Posts a message into a mailbox.

- **msg_t chMBPostI (mailbox_t *mbp, msg_t msg)**
  
  Posts a message into a mailbox.

- **msg_t chMBPostAheadTimeout (mailbox_t *mbp, msg_t msg, sysinterval_t timeout)**
  
  Posts an high priority message into a mailbox.

- **msg_t chMBPostAheadTimeoutS (mailbox_t *mbp, msg_t msg, sysinterval_t timeout)**
  
  Posts an high priority message into a mailbox.

- **msg_t chMBPostAheadI (mailbox_t *mbp, msg_t msg)**
  
  Posts an high priority message into a mailbox.

- **msg_t chMBFetchTimeout (mailbox_t *mbp, msg_t *msgp, sysinterval_t timeout)**
  
  Retrieves a message from a mailbox.

- **msg_t chMBFetchTimeoutS (mailbox_t *mbp, msg_t *msgp, sysinterval_t timeout)**
  
  Retrieves a message from a mailbox.

- **msg_t chMBFetchI (mailbox_t *mbp, msg_t *msgp)**
  
  Retrieves a message from a mailbox.

## 8.13.1 Detailed Description

Mailboxes code.

# 8.14 chmboxes.h File Reference

Mailboxes macros and structures.

## Data Structures

- **struct mailbox_t**
  
  Structure representing a mailbox object.
Macros

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

Functions

- void chMBObjectInit (mailbox_t *mbp, msg_t *buf, size_t n)
  Initializes a mailbox_t object.
- void chMBReset (mailbox_t *mbp)
  Resets a mailbox_t object.
- void chMBResetI (mailbox_t *mbp)
  Resets a mailbox_t object.
- msg_t chMBPostTimeout (mailbox_t *mbp, msg_t msg, sysinterval_t timeout)
  Posts a message into a mailbox.
- msg_t chMBPostTimeoutS (mailbox_t *mbp, msg_t msg, sysinterval_t timeout)
  Posts a message into a mailbox.
- msg_t chMBPostI (mailbox_t *mbp, msg_t msg)
  Posts a message into a mailbox.
- msg_t chMBPostAheadTimeout (mailbox_t *mbp, msg_t msg, sysinterval_t timeout)
  Posts an high priority message into a mailbox.
- msg_t chMBPostAheadTimeoutS (mailbox_t *mbp, msg_t msg, sysinterval_t timeout)
  Posts an high priority message into a mailbox.
- msg_t chMBPostAheadI (mailbox_t *mbp, msg_t msg)
  Posts an high priority message into a mailbox.
- msg_t chMBFetchTimeout (mailbox_t *mbp, msg_t *msgp, sysinterval_t timeout)
  Retrieves a message from a mailbox.
- msg_t chMBFetchTimeoutS (mailbox_t *mbp, msg_t *msgp, sysinterval_t timeout)
  Retrieves a message from a mailbox.
- msg_t 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.

8.14.1 Detailed Description

Mailboxes macros and structures.
8.15 chmemcore.c File Reference

Core memory manager code.

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

Functions

- `void __core_init (void)`
  
  Low level memory manager initialization.

- `void * chCoreAllocFromBaseI (size_t size, unsigned align, size_t offset)`
  
  Allocates a memory block starting from the lowest address upward.

- `void * chCoreAllocFromTopI (size_t size, unsigned align, size_t offset)`
  
  Allocates a memory block starting from the top address downward.

- `void * chCoreAllocFromBase (size_t size, unsigned align, size_t offset)`
  
  Allocates a memory block starting from the lowest address upward.

- `void * chCoreAllocFromTop (size_t size, unsigned align, size_t offset)`
  
  Allocates a memory block starting from the top address downward.

- `size_t chCoreGetStatusX (void)`
  
  Core memory status.

Variables

- `memcore_t ch_memcore`
  
  Memory core descriptor.

8.15.1 Detailed Description

Core memory manager code.

8.16 chmemcore.h File Reference

Core memory manager macros and structures.

Data Structures

- `struct memcore_t`
  
  Type of memory core object.

Macros

- `#define CH_CFG_MEMCORE_SIZE 0`
  
  Managed RAM size.

- `#define chCoreAllocAlignedWithOffsetI chCoreAllocFromTopI`
  
  Allocates a memory block.

- `#define chCoreAllocAlignedWithOffset chCoreAllocFromTop`
  
  Allocates a memory block.
Typedefs

- typedef void *(memgetfunc_t) (size_t size, unsigned align)
  
  Memory get function.

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

Functions

- void __core_init (void)
  
  Low level memory manager initialization.

- void * chCoreAllocFromBaseI (size_t size, unsigned align, size_t offset)
  
  Allocates a memory block starting from the lowest address upward.

- void * chCoreAllocFromTopI (size_t size, unsigned align, size_t offset)
  
  Allocates a memory block starting from the top address downward.

- void * chCoreAllocFromBase (size_t size, unsigned align, size_t offset)
  
  Allocates a memory block starting from the lowest address upward.

- void * chCoreAllocFromTop (size_t size, unsigned align, size_t offset)
  
  Allocates a memory block starting from the top address downward.

- size_t chCoreGetStatusX (void)
  
  Core memory status.

- static void * chCoreAllocAlignedI (size_t size, unsigned align)
  
  Allocates a memory block.

- static void * chCoreAllocAligned (size_t size, unsigned align)
  
  Allocates a memory block.

- static void * chCoreAllocI (size_t size)
  
  Allocates a memory block.

- static void * chCoreAlloc (size_t size)
  
  Allocates a memory block.

8.16.1 Detailed Description

Core memory manager macros and structures.

8.17 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.

8.17.1 Detailed Description

Memory heaps code.

8.18 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.

8.18.1 Detailed Description

Memory heaps macros and structures.

8.19 chmempools.c File Reference

Memory Pools code.

```c
#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.
8.19.1 Detailed Description

Memory Pools code.

8.20 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.

### 8.20.1 Detailed Description

Memory Pools macros and structures.

### 8.21 chmsg.c File Reference

Nil RTOS synchronous messages source file.

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

#### Functions

- **msg_t chMsgSend (thread_t *tp, msg_t msg)**
  Sends a message to the specified thread.

- **thread_t * chMsgWait (void)**
  Suspends the thread and waits for an incoming message.

- **thread_t * chMsgWaitTimeout (sysinterval_t timeout)**
  Suspends the thread and waits for an incoming message or a timeout to occur.

- **thread_t * chMsgWaitTimeoutS (sysinterval_t timeout)**
  Suspends the thread and waits for an incoming message or a timeout to occur.

- **void chMsgRelease (thread_t *tp, msg_t msg)**
  Releases a sender thread specifying a response message.
8.21.1 Detailed Description

Nil RTOS synchronous messages source file.

8.22 chmsg.h File Reference

Nil RTOS synchronous messages header file.

Macros

Macro Functions

- `#define chMsgWaitS() chMsgWaitTimeoutS(TIME_INFINITE)`
  Suspends the thread and waits for an incoming message.
- `#define chMsgGet(tp) ((tp)->sntmsg)`
  Returns the message carried by the specified thread.
- `#define chMsgReleaseS(tp, msg)`
  Releases the thread waiting on top of the messages queue.

Functions

- `msg_t chMsgSend (thread_t *tp, msg_t msg)`
  Sends a message to the specified thread.
- `thread_t * chMsgWait (void)`
  Suspends the thread and waits for an incoming message.
- `thread_t * chMsgWaitTimeout (sysinterval_t timeout)`
  Suspends the thread and waits for an incoming message or a timeout to occur.
- `thread_t * chMsgWaitTimeoutS (sysinterval_t timeout)`
  Suspends the thread and waits for an incoming message or a timeout to occur.
- `void chMsgRelease (thread_t *tp, msg_t msg)`
  Releases a sender thread specifying a response message.

8.22.1 Detailed Description

Nil RTOS synchronous messages header file.

8.23 chobjcaches.c File Reference

Objects Caches code.

#include "ch.h"
Functions

- static oc_object_t * hash_get_s (objects_cache_t *ocp, uint32_t group, uint32_t key)
  Returns an object pointer from the cache, if present.
- static oc_object_t * lru_get_last_s (objects_cache_t *ocp)
  Gets the least recently used object buffer from the LRU list.
- void chCacheObjectInit (objects_cache_t *ocp, ucnt_t hashn, oc_hash_header_t *hashp, ucnt_t objn, size_t objsz, void *objvp, oc_readf_t readf, oc_writef_t writef)
  Initializes a objects_cache_t object.
- oc_object_t * chCacheGetObject (objects_cache_t *ocp, uint32_t group, uint32_t key)
  Retrieves an object from the cache.
- void chCacheReleaseObjectI (objects_cache_t *ocp, oc_object_t *objp)
  Releases an object into the cache.
- bool chCacheReadObject (objects_cache_t *ocp, oc_object_t *objp, bool async)
  Reads object data from the storage.
- bool chCacheWriteObject (objects_cache_t *ocp, oc_object_t *objp, bool async)
  Writes the object data back to storage.

8.23.1 Detailed Description

Objects Caches code.

Objects caches.

Operation mode

An object cache allows to retrieve and release objects from a slow media, for example a disk or flash. The most recently used objects are kept in a series of RAM buffers making access faster. Objects are identified by a pair <group, key> which could be mapped, for example, to a disk drive identifier and sector identifier. Read and write operations are performed using externally-supplied functions, the cache is device-agnostic.

The cache uses internally an hash table, the size of the table should be dimensioned to minimize the risk of hash collisions, a factor of two is usually acceptable, it depends on the specific application requirements. Operations defined for caches:

- Get Object: Retrieves an object from cache, if not present then an empty buffer is returned.
- Read Object: Retrieves an object from cache, if not present a buffer is allocated and the object is read from the media.
- Release Object: Releases an object to the cache handling the media update, if required.

Precondition

In order to use the pipes APIs the CH_CFG_USE_OBJ_CACHES option must be enabled in chconf.h.

Note

Compatible with RT and NIL.
8.24 chobjcaches.h File Reference

Objects Caches macros and structures.

Data Structures

- struct ch_oc_hash_header
  Structure representing an hash table element.
- struct ch_oc_lru_header
  Structure representing an hash table element.
- struct ch_oc_object
  Structure representing a cached object.
- struct ch_objects_cache
  Structure representing a cache object.

Macros

Cached objects flags

- #define OC_FLAG_INLRU 0x00000001U
- #define OC_FLAG_INHASH 0x00000002U
- #define OC_FLAG_SHARED 0x00000004U
- #define OC_FLAG_NOTSYNC 0x00000008U
- #define OC_FLAG_LAZYWRITE 0x00000010U
- #define OC_FLAG_FORGET 0x00000020U

Typedefs

- typedef uint32_t oc_flags_t
  Flags of cached objects.
- typedef struct ch_oc_hash_header oc_hash_header_t
  Type of a hash element header.
- typedef struct ch_oc_lru_header oc_lru_header_t
  Type of an LRU element header.
- typedef struct ch_oc_object oc_object_t
  Type of a cached object.
- typedef struct ch_objects_cache objects_cache_t
  Type of a cache object.
- typedef bool(oc_readf_t) (objects_cache_t *ocp, oc_object_t *objp, bool async)
  Object read function.
- typedef bool(oc_writef_t) (objects_cache_t *ocp, oc_object_t *objp, bool async)
  Object write function.
Functions

- void chCacheObjectInit (objects_cache_t *ocp, ucnt_t hashn, oc_hash_header_t *hashp, ucnt_t objn, size_t objsz, void *objvp, oc_readf_t readf, oc_writef_t writef)
  Initializes an objects_cache_t object.
- oc_object_t * chCacheGetObject (objects_cache_t *ocp, uint32_t group, uint32_t key)
  Retrieves an object from the cache.
- void chCacheReleaseObject (objects_cache_t *ocp, oc_object_t *objp)
  Releases an object into the cache.
- bool chCacheReadObject (objects_cache_t *ocp, oc_object_t *objp, bool async)
  Reads object data from the storage.
- bool chCacheWriteObject (objects_cache_t *ocp, oc_object_t *objp, bool async)
  Writes the object data back to storage.
- static void chCacheReleaseObject (objects_cache_t *ocp, oc_object_t *objp)
  Releases an object into the cache.

8.24.1 Detailed Description

Objects Caches macros and structures.

8.25 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.
Allocates a free object.
- static void chFifoReturnObject (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.

8.25.1 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.

8.26 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.

8.26.1 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.

8.27 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.

8.27.1 Detailed Description

Pipes macros and structures.

8.28 chsem.c File Reference

Nil RTOS semaphores source file.

#include "ch.h"

Functions

- 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 chSemResetWithMessage (semaphore_t *sp, cnt_t n, msg_t msg)
  Performs a reset operation on the semaphore.
- void chSemResetWithMessageI (semaphore_t *sp, cnt_t n, msg_t msg)
  Performs a reset operation on the semaphore.
8.28.1 Detailed Description

Nil RTOS semaphores source file.

8.29 chsem.h File Reference

Nil RTOS semaphores header file.

Macros

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 chSemObjectInit(sp, n) ((sp)->cnt = (n))
  Initializes a semaphore with the specified counter value.
- #define chSemReset(sp, n) chSemResetWithMessage(sp, n, MSG_RESET)
  Performs a reset operation on the semaphore.
- #define chSemReset(sp, n) chSemResetWithMessageI(sp, n, MSG_RESET)
  Performs a reset operation on the semaphore.
- #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.

Functions

- 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 chSemResetWithMessage (semaphore_t *sp, cnt_t n, msg_t msg)
  Performs a reset operation on the semaphore.
- void chSemResetWithMessageI (semaphore_t *sp, cnt_t n, msg_t msg)
  Performs a reset operation on the semaphore.

8.29.1 Detailed Description

Nil RTOS semaphores header file.
Index

EventoSource_Data
  Events, 102
__BseSemaphore_Data
  Binary Semaphores, 131
__chibios_nil__
  Base API, 31
__CH_Stringify
  Base API, 36
__guardedMemoryPool_Data
  Memory Pools, 211
__mailbox_Data
  Mailboxes, 141
__MemoryPool_Data
  Memory Pools, 210
__pipe_Data
  Pipes, 158
__Semaphore_Data
  Semaphores, 89
__Threads_Queue_Data
  Base API, 47
__ch_delegate_fn0
  Delegate Threads, 167
__ch_delegate_fn1
  Delegate Threads, 168
__ch_delegate_fn2
  Delegate Threads, 168
__ch_delegate_fn3
  Delegate Threads, 168
__ch_delegate_fn4
  Delegate Threads, 170
__core_init
  Core Memory Manager, 194
__dbg_check_disable
  Base API, 64
__dbg_check_enable
  Base API, 65
__dbg_check_enter_isr
  Base API, 68
__dbg_check_leave_isr
  Base API, 68
__dbg_check_lock
  Base API, 66
__dbg_check_lock_from_isr
  Base API, 67
__dbg_check_suspend
  Base API, 65
__dbg_check_unlock
  Base API, 66
__dbg_check_unlock_from_isr
  Base API, 67
__factory_init
  Dynamic Objects Factory, 264
__heap_init
  Memory Heaps, 204
__oslib_init
  Version Numbers and Identification, 128

align
  memory_pool_t, 325

All_Events
  Events, 101

Arg
  nil_thread_descriptor, 335

Base API, 24
__chibios_nil__, 31
__CH_stringify, 36
__Threads_Queue_Data, 47
__dbg_check_disable, 64
__dbg_check_enable, 65
__dbg_check_enter_isr, 68
__dbg_check_leave_isr, 68
__dbg_check_lock, 66
__dbg_check_lock_from_isr, 67
__dbg_check_suspend, 65
__dbg_check_unlock, 66
__dbg_check_unlock_from_isr, 67
CH_FAST_IRQ_HANDLER, 42
CH_IRQ_epiLOGUE, 42
CH_IRQ_HANDLER, 42
CH_IRQ_ISVALID_KERNEL_PRIORITY, 41
CH_IRQ_ISVALID_PRIORITY, 40
CH_IRQ_PROLOGUE, 41
CH_KERNEL_MAJOR, 32
CH_KERNEL_MINOR, 32
CH_KERNEL_PATCH, 32
CH_KERNEL_STABLE, 31
CH_KERNEL_VERSION, 32
chDbgAssert, 59
chDbgCheck, 58
chDbgCheckClassI, 69
chDbgCheckClassS, 69
chSchDoPreemption, 76
chSchGoSleep, 50
chSchGoSleepTimeout, 77
chSchIsPreemptionRequired, 76
chSchIsRescRequired, 51
chSchReadyI, 75
chSchRescheduleS, 77
chSchWakeupS, 51
chSysDisable, 48
chSysEnable, 48
chSysGetRealtimeCounterX, 47
chSysGetStatusAndLockX, 73
chSysHalt, 71
chSysInit, 70
chSysIsCounterWithinX, 74
chSysLock, 49
chSysLockFromISR, 49
chSysPolledDelayX, 75
chSysRestoreStatusX, 73
chSysSuspend, 48
chSysTimerHandlerI, 72
chSysUnconditionalLock, 72
chSysUnconditionalUnlock, 72
chSysUnlock, 49
chSysUnlockFromISR, 50
chThdCreate, 80
chThdCreateI, 79
chThdDequeueAllI, 87
chThdDequeueNextI, 86
chThdDoDequeueNextI, 86
chThdEnqueueTimeoutS, 85
chThdExit, 81
chThdGetPriorityX, 52
chThdGetSelfX, 52
chThdQueueEmptyI, 55
chThdQueueObjectInit, 55
chThdResume, 84
chThdResumeI, 83
chThdResumeS, 52
chThdSleep, 84
chThdSleepMicroseconds, 54
chThdSleepMilliseconds, 53
chThdSleepS, 54
chThdSleepSeconds, 53
chThdSleepUntil, 84
chThdSleepUntilS, 54
chThdSuspendTimeoutS, 82
chThdWait, 82
chTimeAddX, 57
chTimeDiffX, 58
chTimelsInRangeX, 78
chVTGetSystemTimeX, 56
chVTIsSystemTimeWithinX, 57
chVTTimeElapsedSinceX, 56
cnt_t, 61
eventflags_t, 61
eventid_t, 60
eventmask_t, 60
FALSE, 32
MEM_ALIGN_MASK, 38
MEM_ALIGN_NEXT, 38
MEM_ALIGN_PREV, 37
MEM_IS_ALIGNED, 38
MEM_IS_VALID_ALIGNMENT, 38
MSG_OK, 33
MSG_RESET, 33
msg_t, 60
MSG_TIMEOUT, 33
nil, 88
nil_find_thread, 63
nil_ready_all, 64
NIL_STATE_FINAL, 36
NIL_STATE_READY, 34
NIL_STATE_SLEEPING, 34
NIL_STATE_SNDSMSGQ, 35
NIL_STATE_SUSPENDED, 34
NIL_STATE_WTANDEVT, 35
NIL_STATE_WTEXIT, 35
NIL_STATE_WTMSC, 35
NIL_STATE_WTOREVT, 35
NIL_STATE_WTQUEUE, 35
NIL_STATE_WTSTART, 34
os_instance_t, 62
rtcnt_t, 59
semaphore_t, 63
stkalign_t, 60
sysinterval_t, 61
syssts_t, 60
systime_t, 61
tfunc_t, 62
THD_FUNCTION, 40
THD_IDLE_BASE, 36
THD_TABLE_BEGIN, 36
THD_TABLE_END, 37
THD_TABLE_THREAD, 36
THD_WORKING_AREA, 39
THD_WORKING_AREA_END, 40
THD_WORKING_AREA_SIZE, 39
thread_descriptor_t, 62
thread_reference_t, 62
thread_t, 62
THREADS_QUEUE_DECL, 47
threads_queue_t, 63
time_conv_t, 61
TIME_I2MS, 45
TIME_I2S, 45
TIME_I2US, 46
TIME_IMMEDIATE, 33
TIME_INFINITE, 33
TIME_MAX_INTERVAL, 34
TIME_MAX_SYSTIME, 34
TIME_MS2I, 43
TIME_S2I, 43
TIME_US2I, 44
tprio_t, 60
TRUE, 32
tstate_t, 60
ucnt_t, 61
basemem
memcore_t, 322
Binary Semaphores, 131
INDEX

_BSEMAPHORE_DATA, 131
binary_semaphore_t, 132
BSEMAPHORE_DECL, 132
chBSemGetStateI, 139
chBSemObjectInit, 132
chBSemReset, 137
chBSemResetI, 136
chBSemSignal, 138
chBSemSignalI, 137
chBSemWait, 133
chBSemWaitS, 133
chBSemWaitTimeout, 135
chBSemWaitTimeoutS, 134
binary_semaphore_t
Binary Semaphores, 132
BSEMAPHORE_DECL
Binary Semaphores, 132
buf_list
ch_objects_factory, 302
buffer
mailbox_t, 320
pipe_t, 338
cache_sem
ch_objects_cache, 299
ch.c, 343
ch.h, 345
ch_binary_semaphore, 283
CH_CFG_AUTOSTART_THREADS
Configuration, 14
CH_CFG_FACTORY_GENERIC_BUFFERS
Configuration, 20
Dynamic Objects Factory, 261
CH_CFG_FACTORY_MAILBOXES
Configuration, 20
Dynamic Objects Factory, 262
CH_CFG_FACTORY_MAX_NAMES_LENGTH
Configuration, 19
Dynamic Objects Factory, 261
CH_CFG_FACTORY_OBJ_FIFOS
Configuration, 20
Dynamic Objects Factory, 262
CH_CFG_FACTORY_OBJECTS_REGISTRY
Configuration, 20
Dynamic Objects Factory, 261
CH_CFG_FACTORY_PIPES
Configuration, 20
Dynamic Objects Factory, 262, 263
CH_CFG_FACTORY_SEMAPHORES
Configuration, 20
Dynamic Objects Factory, 261
CH_CFG_IDLE_ENTER_HOOK
Configuration, 23
CH_CFG_IDLE_LEAVE_HOOK
Configuration, 23
CH_CFG_MAX_THREADS
Configuration, 14
CH_CFG_MEMCORE_SIZE
Configuration, 17

Core Memory Manager, 192
CH_CFG_ST_FREQUENCY
Configuration, 15
CH_CFG_ST_RESOLUTION
Configuration, 14
CH_CFG_ST_TIMDELTA
Configuration, 15
CH_CFG_SYSTEM_HALT_HOOK
Configuration, 23
CH_CFG_SYSTEM_INIT_HOOK
Configuration, 22
CH_CFG_THREAD_EXIT_HOOK
Configuration, 22
CH_CFG_THREAD_EXT_FIELDS
Configuration, 22
CH_CFG_THREAD_EXT_INIT_HOOK
Configuration, 22
CH_CFG_USE_DELEGATES
Configuration, 19
CH_CFG_USE_EVENTS
Configuration, 16
CH_CFG_USE_FACTORY
Configuration, 19
CH_CFG_USE_HEAP
Configuration, 17
CH_CFG_USE_JOBS
Configuration, 19
CH_CFG_USE_MAILBOXES
Configuration, 16
CH_CFG_USE_MEMCORE
Configuration, 17
CH_CFG_USE_MEMPOOLS
Configuration, 18
CH_CFG_USE_MESSAGES
Configuration, 16
CH_CFG_USE_MUTEXES
Configuration, 16
CH_CFG_USE_OBJ_CACHES
Configuration, 18
CH_CFG_USE_OBJ_FIFOS
Configuration, 18
CH_CFG_USE_PIPES
Configuration, 18
CH_CFG_USE_SEMAPHORES
Configuration, 15
CH_CFG_USE_WAITEXIT
Configuration, 15
CH_DBG_ENABLE_ASSERTS
Configuration, 21
CH_DBG_ENABLE_CHECKS
Configuration, 21
CH_DBG_ENABLE_STACK_CHECK
Configuration, 22
CH_DBG_STATISTICS
Configuration, 21
CH_DBG_SYSTEM_STATE_CHECK
Configuration, 21
ch_dyn_element, 284
Binary Semaphores, 136
chBSemSignal
Binary Semaphores, 138
chBSemSignalI
Binary Semaphores, 137
chBSemWait
Binary Semaphores, 133
chBSemWaitS
Binary Semaphores, 133
chBSemWaitTimeout
Binary Semaphores, 135
chBSemWaitTimeoutS
Binary Semaphores, 134
chCacheGetObject
Objects Caches, 253
chCacheObjectInit
Objects Caches, 252
chCacheReadObject
Objects Caches, 254
chCacheReleaseObject
Objects Caches, 256
chCacheReleaseObjectI
Objects Caches, 254
chCacheWriteObject
Objects Caches, 255
chconf.h, 353
chCoreAlloc
Core Memory Manager, 200
chCoreAllocAligned
Core Memory Manager, 199
chCoreAllocAlignedI
Core Memory Manager, 198
chCoreAllocAlignedWithOffset
Core Memory Manager, 193
chCoreAllocAlignedWithOffsetI
Core Memory Manager, 192
chCoreAllocFromBase
Core Memory Manager, 196
chCoreAllocFromBaseI
Core Memory Manager, 194
chCoreAllocFromTop
Core Memory Manager, 197
chCoreAllocFromTopI
Core Memory Manager, 195
chCoreAlloc1
Core Memory Manager, 199
chCoreGetStatusX
Core Memory Manager, 198
chDbgAssert
Base API, 59
chDbgCheck
Base API, 58
chDbgCheckClassI
Base API, 69
chDbgCheckClassS
Base API, 69
chDelegateCallDirect0
Delegate Threads, 172
chDelegateCallDirect1
Delegate Threads, 173
chDelegateCallDirect2
Delegate Threads, 174
chDelegateCallDirect3
Delegate Threads, 175
chDelegateCallDirect4
Delegate Threads, 176
chDelegateCallVeneer
Delegate Threads, 170
chDelegateDispatch
Delegate Threads, 171
chDelegateDispatchTimeout
Delegate Threads, 171
chdelegates.c, 355
chdelegates.h, 356
chevt.c, 357
chevt.h, 358
chEvtAddEvents
Events, 111
chEvtAddEventsI
Events, 105
chEvtBroadcast
Events, 104
chEvtBroadcastFlags
Events, 115
chEvtBroadcastFlagsI
Events, 111
chEvtBroadcast1
Events, 105
chEvtDispatch
Events, 116
chEvtGetAndClearEvents
Events, 110
chEvtGetAndClearEventsI
Events, 110
chEvtGetAndClearFlags
Events, 112
chEvtGetAndClearFlagsI
Events, 115
chEvtGetEventsX
Events, 106
chEvtsListeningI
Events, 104
chEvtObjectInit
Events, 102
chEvtRegister
Events, 104
chEvtRegisterMask
Events, 103
chEvtRegisterMaskWithFlags
Events, 108
chEvtSignal
Events, 112
chEvtSignall
Events, 114
chEvtUnregister
Events, 109
chEvtWaitAll
Events, 107
chEvtWaitAllTimeout
Events, 118
chEvtWaitAny
Events, 107
chEvtWaitAnyTimeout
Events, 117
chEvtWaitOne
Events, 106
chEvtWaitOneTimeout
Events, 116
chfactory.c, 360
cchactory.h, 361
chFactoryCreateBuffer
Dynamic Objects Factory, 267
chFactoryCreateMailbox
Dynamic Objects Factory, 271
chFactoryCreateObjectsFIFO
Dynamic Objects Factory, 273
chFactoryCreatePipe
Dynamic Objects Factory, 275
chFactoryCreateSemaphore
Dynamic Objects Factory, 270
chFactoryDuplicateReference
Dynamic Objects Factory, 276
chFactoryFindBuffer
Dynamic Objects Factory, 268
chFactoryFindMailbox
Dynamic Objects Factory, 272
chFactoryFindObject
Dynamic Objects Factory, 265
chFactoryFindObjectByPointer
Dynamic Objects Factory, 266
chFactoryFindObjectFIFO
Dynamic Objects Factory, 274
chFactoryFindPipe
Dynamic Objects Factory, 275
chFactoryFindSemaphore
Dynamic Objects Factory, 270
chFactoryGetBuffer
Dynamic Objects Factory, 279
chFactoryGetBufferSize
Dynamic Objects Factory, 277
chFactoryGetMailbox
Dynamic Objects Factory, 280
chFactoryGetObject
Dynamic Objects Factory, 277
chFactoryGetObjectsFIFO
Dynamic Objects Factory, 280
chFactoryGetPipe
Dynamic Objects Factory, 281
chFactoryGetSemaphore
Dynamic Objects Factory, 279
chFactoryRegisterObject
Dynamic Objects Factory, 265
chFactoryReleaseBuffer
Dynamic Objects Factory, 268
chFactoryReleaseMailbox
Dynamic Objects Factory, 273
chFactoryReleaseObject
Dynamic Objects Factory, 267
chFactoryReleaseObjectsFIFO
Dynamic Objects Factory, 274
chFactoryReleasePipe
Dynamic Objects Factory, 276
chFactoryReleaseSemaphore
Dynamic Objects Factory, 271
chFifoObjectInit
Objects FIFOs, 235
chFifoObjectInitAligned
Objects FIFOs, 234
chFifoReceiveObject
Objects FIFOs, 245
chFifoReceiveObjectTimeout
Objects FIFOs, 247
chFifoReceiveObjectTimeoutS
Objects FIFOs, 246
chFifoReturnObject
Objects FIFOs, 240
chFifoReturnObjectAlign
Objects FIFOs, 239
chFifoReturnObjectS
Objects FIFOs, 239
chFifoSendObject
Objects FIFOs, 242
chFifoSendObjectAhead
Objects FIFOs, 245
chFifoSendObjectAheadS
Objects FIFOs, 243
chFifoSendObjectAheadS
Objects FIFOs, 242
chFifoSendObjectAlign
Objects FIFOs, 244
chFifoSendObjectS
Objects FIFOs, 241
chFifoSendObjectS
Objects FIFOs, 242
chFifoTakeObject
Objects FIFOs, 236
chFifoTakeObjectTimeout
Objects FIFOs, 238
chFifoTakeObjectTimeoutS
Objects FIFOs, 237
chGuardedPoolAdd
Memory Pools, 228
chGuardedPoolAddI
Memory Pools, 229
chGuardedPoolAddS
Memory Pools, 230
chGuardedPoolAlloc
Memory Pools, 225
chGuardedPoolAllocTimeout
Memory Pools, 220
chGuardedPoolAllocTimeoutS
Memory Pools, 219
chGuardedPoolFree
Memory Pools, 221
INDEX 389

chGuardedPoolFreeI
Memory Pools, 226
chGuardedPoolFreeS
Memory Pools, 227
chGuardedPoolGetCounterI
Memory Pools, 225
chGuardedPoolLoadArray
Memory Pools, 218
chGuardedPoolObjectInit
Memory Pools, 224
chGuardedPoolObjectInitAligned
Memory Pools, 217
chHeapAlloc
Memory Heaps, 206
chHeapAllocAligned
Memory Heaps, 205
chHeapFree
Memory Heaps, 205
chHeapGetSize
Memory Heaps, 207
chHeapObjectInit
Memory Heaps, 204
chHeapStatus
Memory Heaps, 206
chJobDispatch
Jobs Queues, 188
chJobDispatchTimeout
Jobs Queues, 188
chJobGet
Jobs Queues, 180
chJobGetI
Jobs Queues, 181
chJobGetTimeout
Jobs Queues, 182
chJobGetTimeoutS
Jobs Queues, 182
chJobObjectInit
Jobs Queues, 179
chJobPost
Jobs Queues, 185
chJobPostAhead
Jobs Queues, 187
chJobPostAheadI
Jobs Queues, 186
chJobPostAheadS
Jobs Queues, 186
chJobPostI
Jobs Queues, 183
chJobPostS
Jobs Queues, 184
chjobs.h, 364
chlib.h, 366
chMBFetchI
Mailboxes, 153
chMBFetchTimeout
Mailboxes, 151
chMBFetchTimeoutS
Mailboxes, 152
chMBGetFreeCountI
Mailboxes, 155
chMBGetObjectInit
Mailboxes, 142
chmboxes.c, 367
chmboxes.h, 367
chMBGetI
Mailboxes, 156
chMBPostAheadI
Mailboxes, 150
chMBPostAheadTimeout
Mailboxes, 148
chMBPostAheadTimeoutS
Mailboxes, 149
chMBPostI
Mailboxes, 147
chMBPostTimeout
Mailboxes, 145
chMBPostTimeoutS
Mailboxes, 146
chMBReset
Mailboxes, 143
chMBResetI
Mailboxes, 144
chMBResumeX
Mailboxes, 156
chmemcore.c, 369
chmemcore.h, 369
chmemheaps.c, 370
chmemheaps.h, 371
chmempools.c, 372
chmempools.h, 373
chmsg.c, 374
chmsg.h, 375
chMsgGet
Synchronous Messages, 120
chMsgRelease
Synchronous Messages, 125
chMsgReleaseS
Synchronous Messages, 121
chMsgSend
Synchronous Messages, 122
chMsgWait
Synchronous Messages, 122
chMsgWaitS
Synchronous Messages, 120
chMsgWaitTimeout
Synchronous Messages, 123
chMsgWaitTimeoutS
Synchronous Messages, 124
chobjcaches.c, 375
chobjcaches.h, 377
chobjfifos.h, 378
chPipeGetFreeCount

ChibiOS/NIL
Pipes, 164
chPipeGetSize
Pipes, 163
chPipeGetUsedCount
Pipes, 163
chPipeObjectInit
Pipes, 161
chPipeReadTimeout
Pipes, 162
chPipeReset
Pipes, 161
chPipeResume
Pipes, 164
chpipes.c, 379
chpipes.h, 380
chPipeWriteTimeout
Pipes, 161
chPoolAdd
Memory Pools, 222
chPoolAddI
Memory Pools, 223
chPoolAlloc
Memory Pools, 215
chPoolAllocI
Memory Pools, 214
chPoolFree
Memory Pools, 217
chPoolFreeI
Memory Pools, 216
chPoolLoadArray
Memory Pools, 213
chPoolObjectInit
Memory Pools, 222
chPoolObjectInitAligned
Memory Pools, 212
chSchDoPreemption
Base API, 76
chSchGoSleepS
Base API, 50
chSchGoSleepTimeoutS
Base API, 77
chSchIsPreemptionRequired
Base API, 76
chSchIsRescRequiredI
Base API, 51
chSchReadyI
Base API, 75
chSchRescheduleS
Base API, 77
chSchWakeupS
Base API, 51
chsem.c, 381
chsem.h, 382
chSemFastSignal
Semaphores, 93
chSemFastWait
Semaphores, 93
chSemGetCounterI
Semaphores, 94
chSemObjectInit
Semaphores, 90
chSemReset
Semaphores, 91
chSemResetI
Semaphores, 91
chSemResetWithMessage
Semaphores, 97
chSemResetWithMessageI
Semaphores, 98
chSemSignal
Semaphores, 96
chSemSignalI
Semaphores, 96
chSemWait
Semaphores, 92
chSemWaitS
Semaphores, 92
chSemWaitTimeout
Semaphores, 94
chSemWaitTimeoutS
Semaphores, 95
chSysDisable
Base API, 48
chSysEnable
Base API, 48
chSysGetRealtimeCounterX
Base API, 47
chSysGetStatusAndLockX
Base API, 73
chSysHalt
Base API, 71
chSysInit
Base API, 70
chSysIsCounterWithinX
Base API, 74
chSysLock
Base API, 49
chSysLockFromISR
Base API, 49
chSysPollDelayX
Base API, 75
chSysRestoreStatusX
Base API, 73
chSysSuspend
Base API, 48
chSysTimerHandlerI
Base API, 72
chSysUnconditionalLock
Base API, 72
chSysUnconditionalUnlock
Base API, 72
chSysUnlock
Base API, 49
chSysUnlockFromISR
Base API, 50
chThdCreate
INDEX

Base API, 80
chThdCreatel
  Base API, 79
chThdDequeueAll
  Base API, 87
chThdDequeueNextl
  Base API, 86
chThdEnqueueTimeoutS
  Base API, 85
chThdExit
  Base API, 81
chThdGetPriorityX
  Base API, 52
chThdGetSelfX
  Base API, 52
chThdQueueIsEmptyI
  Base API, 55
chThdQueueObjectInit
  Base API, 55
chThdResume
  Base API, 84
chThdResumeI
  Base API, 83
chThdResumeS
  Base API, 52
chThdSleep
  Base API, 84
chThdSleepMicroseconds
  Base API, 54
chThdSleepMilliseconds
  Base API, 53
chThdSleepS
  Base API, 54
chThdSleepSeconds
  Base API, 53
chThdSleepUntil
  Base API, 84
chThdSleepUntilIS
  Base API, 54
chThdSuspendTimeoutS
  Base API, 82
chThdWait
  Base API, 82
chTimeAddX
  Base API, 57
chTimeDiffX
  Base API, 58
chTimeIsInRangeX
  Base API, 78
chVTGetSystemTimeX
  Base API, 56
chVTsSystemTimeWithinX
  Base API, 57
chVTTimeElapsedSinceX
  Base API, 56
cmtx
  pipe_t, 339
cnt
  mailbox_t, 320
il_threads_queue, 336
pipe_t, 339
cnt_t
  Base API, 61
Complex Services, 232
Configuration, 12
  CH_CFG_AUTOSTART_THREADS, 14
  CH_CFG_FACTORY_GENERIC_BUFFERS, 20
  CH_CFG_FACTORY_MAILBOXES, 20
  CH_CFG_FACTORY_MAX_NAMES_LENGTH, 19
  CH_CFG_FACTORY_OBJ_FIFOS, 20
  CH_CFG_FACTORY_OBJECTS_REGISTRY, 20
  CH_CFG_FACTORY_PIPES, 20
  CH_CFG_FACTORY_SEMAPHORES, 20
  CH_CFG_IDLE_ENTER_HOOK, 23
  CH_CFG_IDLE_LEAVE_HOOK, 23
  CH_CFG_MAX_THREADS, 14
  CH_CFG_MEMCORE_SIZE, 17
  CH_CFG_ST_FREQUENCY, 15
  CH_CFG_ST_RESOLUTION, 14
  CH_CFG_ST_TIMEDELTA, 15
  CH_CFG_SYSTEM_HALT_HOOK, 23
  CH_CFG_SYSTEM_INIT_HOOK, 22
  CH_CFG_THREAD_EXIT_HOOK, 22
  CH_CFG_THREAD_EXT_FIELDS, 22
  CH_CFG_THREAD_EXT_INIT_HOOK, 22
  CH_CFG_USE_DELEGATES, 19
  CH_CFG_USE_EVENTS, 16
  CH_CFG_USE_FACTORY, 19
  CH_CFG_USE_HEAP, 17
  CH_CFG_USE_JOBS, 19
  CH_CFG_USE_MAILBOXES, 16
  CH_CFG_USE_MEMCORE, 17
  CH_CFG_USE_MEMPOOLS, 18
  CH_CFG_USE_MESSAGES, 16
  CH_CFG_USE_MUTEXES, 16
  CH_CFG_USE_OBJ_CACHES, 18
  CH_CFG_USE_OBJ_FIFOS, 18
  CH_CFG_USE_PIPES, 18
  CH_CFG_USE_SEMAPHORES, 15
  CH_CFG_USE_WAITEXIT, 15
  CH_DBG_ENABLE_ASSERTS, 21
  CH_DBG_ENABLE_CHECKS, 21
  CH_DBG_ENABLE_STACK_CHECK, 22
  CHDBG_ENABLE.Assertions, 21
  CH_DBG_ENABLE_STACK_CHECK, 22
  CH_DBG_STATISTICS, 21
  CH_DBG_SYSTEM_STATE_CHECK, 21
Core Memory Manager, 191
  __core_init, 194
  CH_CFG_MEMCORE_SIZE, 192
  ch_memcore, 201
  chCoreAlloc, 200
  chCoreAllocAligned, 199
  chCoreAllocAlignedI, 198
  chCoreAllocAlignedWithOffset, 193
  chCoreAllocAlignedWithOffsetI, 192
ChibiOS/NIL
chCoreAllocFromBase, 196
chCoreAllocFromBaseI, 194
chCoreAllocFromTop, 197
chCoreAllocFromTopI, 195
chCoreAllocI, 199
chCoreGetStatusX, 198

memgetfunc2_t, 193
memgetfunc_t, 193
cctx
nil_thread, 330
current
nil_os_instance, 327
dbg_panic_msg
nil_os_instance, 328
default_heap
Memory Heaps, 208
Delegate Threads, 166
__ch_delegate_fn0, 167
__ch_delegate_fn1, 168
__ch_delegate_fn2, 168
__ch_delegate_fn3, 168
__ch_delegate_fn4, 170
chDelegateCallDirect0, 172
chDelegateCallDirect1, 173
chDelegateCallDirect2, 174
chDelegateCallDirect3, 175
chDelegateCallDirect4, 176
chDelegateCallVeneer, 170
chDelegateDispatch, 171
chDelegateDispatchTimeout, 171
delegate_fn0_t, 167
delegate_fn1_t, 167
delegate_fn2_t, 167
delegate_fn3_t, 167
delegate_fn4_t, 167
delegate_veneer_t, 167
deploy_fn0_t
Delegate Threads, 167
deploy_fn1_t
Delegate Threads, 167
deploy_fn2_t
Delegate Threads, 167
deploy_fn3_t
Delegate Threads, 167
deploy_fn4_t
Delegate Threads, 167
deploy_veneer_t
Delegate Threads, 167
dptr
ch_oc_object, 310
dyn_buffer_t
Dynamic Objects Factory, 263
dyn_element_t
Dynamic Objects Factory, 263
dyn_list_t
Dynamic Objects Factory, 263
dyn_mailbox_t
Dynamic Objects Factory, 264
dyn_objects_fifo_t
Dynamic Objects Factory, 264
dyn_pipe_t
Dynamic Objects Factory, 264
dyn_semaphore_t
Dynamic Objects Factory, 263
Dynamic Objects Factory, 258
__factory_init, 264
CH_CFG_FACTORY_GENERIC_BUFFERS, 261
CH_CFG_FACTORY_MAILBOXES, 262
CH_CFG_FACTORY_MAX_NAMES_LENGTH, 261
CH_CFG_FACTORY_OBJ_FIFO, 267
CH_CFG_FACTORY_OBJECTS_REGISTRY, 261
CH_CFG_FACTORY_PIPES, 262, 263
CH_CFG_FACTORY_SEMAPHORES, 261
ch_factory, 281
chFactoryCreateBuffer, 267
chFactoryCreateMailbox, 271
chFactoryCreateObjectsFIFO, 273
chFactoryCreatePipe, 275
chFactoryCreateSemaphore, 270
chFactoryDuplicateReference, 276
chFactoryFindBuffer, 268
chFactoryFindMailbox, 272
chFactoryFindObject, 265
chFactoryFindObjectByPointer, 266
chFactoryFindObjectByObject, 274
chFactoryFindPipe, 275
chFactoryFindSemaphore, 270
chFactoryGetBuffer, 279
chFactoryGetBufferSize, 277
chFactoryGetMailbox, 280
chFactoryGetObject, 277
chFactoryGetObjectsFIFO, 280
chFactoryGetPipe, 281
chFactoryGetSemaphore, 279
chFactoryRegisterObject, 265
chFactoryReleaseBuffer, 268
chFactoryReleaseMailbox, 273
chFactoryReleaseObject, 267
chFactoryReleaseObjectsFIFO, 274
chFactoryReleasePipe, 276
chFactoryReleaseSemaphore, 271
dyn_buffer_t, 263
dyn_element_t, 263
dyn_list_t, 263
dyn_mailbox_t, 264
dyn_objects_fifo_t, 264
dyn_pipe_t, 264
dyn_semaphore_t, 263
objects_factory_t, 263
registered_object_t, 263
element
ch_dyn_mailbox, 287
ch_dyn_object, 288
ch_dyn_objects_fifo, 289
ch_dyn_pipe, 292

ChibiOS/NIL
ch_dyn_semaphore, 293
ch_registered_static_object, 311

epmask
nil_thread, 332
event_listener, 312
events, 313
flags, 313
listener, 313
next, 313
wflags, 314

EVENT_MASK
Events, 102
event_source, 314
next, 315
event_source_t
Events, 108
eventflags_t
Base API, 61
eventid_t
Base API, 60
eventmask_t
Base API, 60

Events, 100

_EVENTSOURCES_DATA, 102
ALL_EVENTS, 101
chEvtAddEvents, 111
chEvtAddEventsI, 105
chEvtBroadcast, 104
chEvtBroadcastFlags, 115
chEvtBroadcastFlagsI, 111
chEvtBroadcastI, 105
chEvtDispatch, 116
chEvtGetAndClearEvents, 110
chEvtGetAndClearEventsI, 110
chEvtGetAndClearFlags, 112
chEvtGetAndClearFlagsI, 115
chEvtGetEventsX, 106
chEvtsListeningI, 104
chEventListener, 102
chEvtRegister, 104
chEvtRegisterMask, 103
chEvtRegisterMaskWithFlags, 108
chEvtSignal, 112
chEvtSignalI, 114
chEvtUnregister, 109
chEvtWaitAll, 107
chEvtWaitAllTimeout, 118
chEvtWaitAny, 107
chEvtWaitAnyTimeout, 117
chEvtWaitOne, 106
chEvtWaitOneTimeout, 116

EVENT_MASK, 102
event_source_t, 108
EVENTSOURCE_DECL, 102
evhandler_t, 108

Events, 102
evhandler_t
Events, 108
ewmask
nil_thread, 332
FALSE
Base API, 32
fifo
ch_dyn_objects_fifo, 290
fifo_list
ch_objects_factory, 302
flags
event_listener, 313
free
ch_jobs_queue, 295
ch_objects_fifo, 304
funcp
nil_thread_descriptor, 335

guarded_memory_pool_t, 316
pool, 317
sem, 317
GUARDEDMEMORYPOOL_DECL
Memory Pools, 212

hash_get_s
Objects Caches, 251
hash_next
ch_oc_hash_header, 305
ch_oc_lru_header, 307
ch_oc_object, 309
hash_prev
ch_oc_hash_header, 306
ch_oc_lru_header, 307
ch_oc_object, 309

hashn
ch_objects_cache, 298
hashp
ch_objects_cache, 298
header
memory_heap, 323
heap
heap_header, 318
heap_header, 317
heap, 318
next, 318
pages, 318
size, 318
heap_header_t
Memory Heaps, 204

isr_cnt
nil_os_instance, 328

job_descriptor_t
Jobs Queues, 179
job_function_t
Jobs Queues, 179
INDEX

jobarg
  ch_job_descriptor, 294
jobfunc
  ch_job_descriptor, 294
Jobs Queues, 178
  chJobDispatch, 188
  chJobDispatchTimeout, 188
  chJobGet, 180
  chJobGetI, 181
  chJobGetTimeout, 182
  chJobGetTimeoutS, 182
  chJobObjectInit, 179
  chJobPost, 185
  chJobPostAhead, 187
  chJobPostAheadI, 186
  chJobPostAheadS, 186
  chJobPostI, 183
  chJobPostS, 184
job_descriptor_t, 179
job_function_t, 179
jobs_queue_t, 179
  MSG_JOB_NULL, 179
jobs_queue_t
  Jobs Queues, 179
lasttime
  nil_os_instance, 328
listener
  event_listener, 313
lock_cnt
  nil_os_instance, 328
lru
  ch_objects_cache, 299
  ch_objects_cache, 299
  lru_get_last_s
    Objects Caches, 252
lru_next
  ch_oc_lru_header, 307
  ch_oc_object, 309
lru_prev
  ch_oc_lru_header, 307
  ch_oc_object, 309
lru_sem
  ch_objects_cache, 299
MAILBOX_DECL
  Mailboxes, 142
mailbox_t, 319
  buffer, 320
cnt, 320
qr, 321
qw, 321
rdptr, 320
reset, 321
top, 320
wrptr, 320
Mailboxes, 140
  __MAILBOX_DATA, 141
chMBFetchI, 153
chMBFetchTimeout, 151
chMBFetchTimeoutS, 152
chMBGetFreeCountI, 155
chMBGetSizeI, 154
chMBGetUsedCountI, 154
chMOBJECTInit, 142
chMPeekI, 156
chMPopAheadI, 150
chMBPostAheadTimeout, 148
chMBPostAheadTimeoutS, 149
chMBPostI, 147
chMBPostTimeout, 145
chMBPostTimeoutS, 146
chMRestet, 143
chMRestet, 144
chMResumeX, 156
MAILBOX_DECL, 142
mbx
  ch_dyn_mailbox, 287
  ch_jobs_queue, 296
  ch_objects_fifo, 304
mbx_list
  ch_objects_factory, 302
MEM_ALIGN_MASK
  Base API, 37
MEM_ALIGN_NEXT
  Base API, 38
MEM_ALIGN_PREV
  Base API, 37
MEM_ISAligned
  Base API, 38
MEM_ISVALID_ALIGNMENT
  Base API, 38
memcore_t, 321
  basemem, 322
topmem, 322
memgetfunc2_t
  Core Memory Manager, 193
memgetfunc_t
  Core Memory Manager, 193
Memory Heaps, 202
  __heap_init, 204
  CH_HEAP_ALIGNMENT, 203
  CH_HEAP_AREA, 203
  chHeapAlloc, 206
  chHeapAllocAligned, 205
  chHeapFree, 205
  chHeapGetSize, 207
  chHeapObjectInit, 204
  chHeapStatus, 206
default_heap, 208
  heap_header_t, 204
  memory_heap_t, 204
Memory Management, 190
Memory Pools, 209
  __GUARDEDMEMORYPOOL_DATA, 211
  __MEMORYPOOL_DATA, 210
  chGuardedPoolAdd, 228
  chGuardedPoolAddI, 229
chGuardedPoolAddS, 230
chGuardedPoolAlloc, 225
chGuardedPoolAllocTimeout, 220
chGuardedPoolAllocTimeoutS, 219
chGuardedPoolFree, 221
chGuardedPoolFreeI, 226
chGuardedPoolFreeS, 227
chGuardedPoolGetCounterI, 225
chGuardedPoolLoadArray, 218
chGuardedPoolObjectInit, 224
chGuardedPoolObjectInitAligned, 217
chPoolAdd, 222
chPoolAddI, 223
chPoolAlloc, 215
chPoolAllocI, 214
chPoolFree, 217
chPoolFreeI, 216
chPoolLoadArray, 213
chPoolObjectInit, 222
chPoolObjectInitAligned, 212
GUARDEDMEMORYPOOL_DECL, 212
MEMORYPOOL_DECL, 211
memory_heap, 322
header, 323
mtx, 324
provider, 323
memory_heap_t
Memory Heaps, 204
memory_pool_t, 324
align, 325
next, 325
object_size, 325
provider, 325
MEMORYPOOL_DECL
Memory Pools, 211
msg
nil_thread, 331
MSG_JOB_NULL
Jobs Queues, 179
MSG_OK
Base API, 33
MSG_RESET
Base API, 33
msg_t
Base API, 60
MSG_TIMEOUT
Base API, 33
mtx
ch_objects_factory, 301
memory_heap, 324
name
nil_thread_descriptor, 334
next
ch_dyn_element, 285
event_listener, 313
event_source, 315
heap_header, 318
memory_pool_t, 325
nil_os_instance, 327
pool_header, 341
nexttime
nil_os_instance, 328
hil
Base API, 88
NIL Kernel, 11
NIL_find_thread
Base API, 63
nil_os_instance, 326
current, 327
dbg_panic_msg, 328
isr_cnt, 328
lasttime, 328
lock_cnt, 328
next, 327
nexttime, 328
systime, 327
threads, 328
nil_ready_all
Base API, 64
NIL_STATE_FINAL
Base API, 36
NIL_STATE_READY
Base API, 34
NIL_STATE_SLEEPING
Base API, 34
NIL_STATE_SNDMSGQ
Base API, 35
NIL_STATE_SUSPENDED
Base API, 34
NIL_STATE_WTANDEVT
Base API, 35
NIL_STATE_WTEXIT
Base API, 35
NIL_STATE_WTMSG
Base API, 35
NIL_STATE_WTOREVT
Base API, 35
NIL_STATE_WTQUEUE
Base API, 35
NIL_STATE_WTSTART
Base API, 34
nil_thread, 329
cxl, 330
epmask, 332
ewmask, 332
msg, 331
nsp, 331
p, 331
semp, 332
sntmsg, 332
state, 331
timeout, 332
tp, 332
tqp, 331
trp, 331
wabase, 333
INDEX

__PIPE_DATA, 158
chPipeGetFreeCount, 164
chPipeGetSize, 163
chPipeGetUsedCount, 163
chPipeObjectInit, 161
chPipeReadTimeout, 162
chPipeReset, 161
chPipeResume, 164
chPipeWriteTimeout, 161
PIPE_DECL, 159
pipe_read, 160
pipe_write, 159
pool
  guarded_memory_pool_t, 317
pool_header, 340
  next, 341
prio
  nil_thread_descriptor, 334
provider
  memory_heap, 323
  memory_pool_t, 325
qr
  mailbox_t, 321
qw
  mailbox_t, 321
rdptr
  mailbox_t, 320
  pipe_t, 339
readf
  ch_objects_cache, 299
refs
  ch_dyn_element, 285
registered_object_t
  Dynamic Objects Factory, 263
reset
  mailbox_t, 321
  pipe_t, 339
rmtx
  pipe_t, 340
rtcnt_t
  Base API, 59
rtr
  pipe_t, 339
sem
  ch_dyn_semaphore, 293
  guarded_memory_pool_t, 317
sem_list
  ch_objects_factory, 302
sem_pool
  ch_objects_factory, 302
SEMAPHORE_DECL
  Semaphores, 90
semaphore_t
  Base API, 63
Semaphores, 89
  __SEMAPHORE_DATA, 89
  chSemFastSignall, 93
  chSemFastWaitl, 93
  chSemGetCounterl, 94
  chSemObjectInit, 90
  chSemReset, 91
  chSemResetl, 91
  chSemResetWithMessage, 97
  chSemResetWithMessageI, 98
  chSemSignal, 96
  chSemSignall, 96
  chSemWait, 92
  chSemWaitS, 92
  chSemWaitTimeout, 94
  chSemWaitTimeoutS, 95
SEMAPHORE_DECL, 90
semp
  nil_thread, 332
size
  heap_header, 318
sntmsg
  nil_thread, 332
state
  nil_thread, 331
stkalign_t
  Base API, 60
Synchronization, 130
Synchronous Messages, 120
  chMsgGet, 120
  chMsgRelease, 125
  chMsgReleaseS, 121
  chMsgSend, 122
  chMsgWait, 122
  chMsgWaitS, 120
  chMsgWaitTimeout, 123
  chMsgWaitTimeoutS, 124
sysinterval_t
  Base API, 61
syssts_t
  Base API, 60
systime
  nil_os_instance, 327
systime_t
  Base API, 61
tfunc_t
  Base API, 62
THD_FUNCTION
  Base API, 40
THD_IDLE_BASE
  Base API, 36
THD_TABLE_BEGIN
  Base API, 36
THD_TABLE_END
  Base API, 37
THD_TABLE_THREAD
  Base API, 36
THD_WORKING_AREA
  Base API, 39
THD_WORKING_AREA_END

ChibiOS/NIL