<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.8.2.12 CH_CFG_USE_SEMAPHORES</td>
<td>CH_CFG_USE_SEMAPHORES</td>
<td>38</td>
</tr>
<tr>
<td>7.8.2.13 CH_CFG_USE_SEMAPHORES_PRIORITY</td>
<td>CH_CFG_USE_SEMAPHORES_PRIORITY</td>
<td>38</td>
</tr>
<tr>
<td>7.8.2.14 CH_CFG_USE_MUTEXES</td>
<td>CH_CFG_USE_MUTEXES</td>
<td>38</td>
</tr>
<tr>
<td>7.8.2.15 CH_CFG_USE_MUTEXES_RECURSIVE</td>
<td>CH_CFG_USE_MUTEXES_RECURSIVE</td>
<td>38</td>
</tr>
<tr>
<td>7.8.2.16 CH_CFG_USE_CONDVARS</td>
<td>CH_CFG_USE_CONDVARS</td>
<td>39</td>
</tr>
<tr>
<td>7.8.2.17 CH_CFG_USE_CONDVARS_TIMEOUT</td>
<td>CH_CFG_USE_CONDVARS_TIMEOUT</td>
<td>39</td>
</tr>
<tr>
<td>7.8.2.18 CH_CFG_USE_EVENTS</td>
<td>CH_CFG_USE_EVENTS</td>
<td>39</td>
</tr>
<tr>
<td>7.8.2.19 CH_CFG_USE_EVENTS_TIMEOUT</td>
<td>CH_CFG_USE_EVENTS_TIMEOUT</td>
<td>40</td>
</tr>
<tr>
<td>7.8.2.20 CH_CFG_USE_MESSAGES</td>
<td>CH_CFG_USE_MESSAGES</td>
<td>40</td>
</tr>
<tr>
<td>7.8.2.21 CH_CFG_USE_MESSAGES_PRIORITY</td>
<td>CH_CFG_USE_MESSAGES_PRIORITY</td>
<td>40</td>
</tr>
<tr>
<td>7.8.2.22 CH_CFG_USE_DYNAMIC</td>
<td>CH_CFG_USE_DYNAMIC</td>
<td>41</td>
</tr>
<tr>
<td>7.8.2.23 CH_CFG_USE_MAILBOXES</td>
<td>CH_CFG_USE_MAILBOXES</td>
<td>41</td>
</tr>
<tr>
<td>7.8.2.24 CH_CFG_USE_MEMCORE</td>
<td>CH_CFG_USE_MEMCORE</td>
<td>41</td>
</tr>
<tr>
<td>7.8.2.25 CH_CFG_MEMCORE_SIZE</td>
<td>CH_CFG_MEMCORE_SIZE</td>
<td>42</td>
</tr>
<tr>
<td>7.8.2.26 CH_CFG_USE_HEAP</td>
<td>CH_CFG_USE_HEAP</td>
<td>42</td>
</tr>
<tr>
<td>7.8.2.27 CH_CFG_USE_MEMPOOLS</td>
<td>CH_CFG_USE_MEMPOOLS</td>
<td>42</td>
</tr>
<tr>
<td>7.8.2.28 CH_CFG_USE_OBJ_FIFOS</td>
<td>CH_CFG_USE_OBJ_FIFOS</td>
<td>43</td>
</tr>
<tr>
<td>7.8.2.29 CH_CFG_USEPIPES</td>
<td>CH_CFG_USEPIPES</td>
<td>43</td>
</tr>
<tr>
<td>7.8.2.30 CH_CFG_USE_OBJ_CACHES</td>
<td>CH_CFG_USE_OBJ_CACHES</td>
<td>43</td>
</tr>
<tr>
<td>7.8.2.31 CH_CFG_USE_DELEGATES</td>
<td>CH_CFG_USE_DELEGATES</td>
<td>43</td>
</tr>
<tr>
<td>7.8.2.32 CH_CFG_USE_JOBS</td>
<td>CH_CFG_USE_JOBS</td>
<td>44</td>
</tr>
<tr>
<td>7.8.2.33 CH_CFG_USE_FACTORY</td>
<td>CH_CFG_USE_FACTORY</td>
<td>44</td>
</tr>
<tr>
<td>7.8.2.34 CH_CFG_FACTORY_MAX_NAMES_LENGTH</td>
<td>CH_CFG_FACTORY_MAX_NAMES_LENGTH</td>
<td>44</td>
</tr>
<tr>
<td>7.8.2.35 CH_CFG_FACTORY_OBJECTS_REGISTRY</td>
<td>CH_CFG_FACTORY_OBJECTS_REGISTRY</td>
<td>44</td>
</tr>
<tr>
<td>7.8.2.36 CH_CFG_FACTORY_GENERIC_BUFFERS</td>
<td>CH_CFG_FACTORY_GENERIC_BUFFERS</td>
<td>44</td>
</tr>
<tr>
<td>7.8.2.37 CH_CFG_FACTORY_SEMAPHORES</td>
<td>CH_CFG_FACTORY_SEMAPHORES</td>
<td>45</td>
</tr>
<tr>
<td>7.8.2.38 CH_CFG_FACTORY_MAILBOXES</td>
<td>CH_CFG_FACTORY_MAILBOXES</td>
<td>45</td>
</tr>
<tr>
<td>7.8.2.39 CH_CFG_FACTORY_OBJ_FIFOS</td>
<td>CH_CFG_FACTORY_OBJ_FIFOS</td>
<td>45</td>
</tr>
<tr>
<td>7.8.2.40 CH_CFG_FACTORY_PIPES</td>
<td>CH_CFG_FACTORY_PIPES</td>
<td>45</td>
</tr>
<tr>
<td>7.8.2.41 CH_DBG_STATISTICS</td>
<td>CH_DBG_STATISTICS</td>
<td>45</td>
</tr>
<tr>
<td>7.8.2.42 CH_DBG_SYSTEM_STATE_CHECK</td>
<td>CH_DBG_SYSTEM_STATE_CHECK</td>
<td>46</td>
</tr>
<tr>
<td>7.8.2.43 CH_DBG_ENABLE_CHECKS</td>
<td>CH_DBG_ENABLE_CHECKS</td>
<td>46</td>
</tr>
<tr>
<td>7.8.2.44 CH_DBG_ENABLE_ASSERTS</td>
<td>CH_DBG_ENABLE_ASSERTS</td>
<td>46</td>
</tr>
<tr>
<td>7.8.2.45 CH_DBGTRACE_MASK</td>
<td>CH_DBGTRACE_MASK</td>
<td>46</td>
</tr>
<tr>
<td>7.8.2.46 CH_DBGTRACE_BUFFER_SIZE</td>
<td>CH_DBGTRACE_BUFFER_SIZE</td>
<td>47</td>
</tr>
<tr>
<td>7.8.2.47 CH_DBG_ENABLE_STACK_CHECK</td>
<td>CH_DBG_ENABLE_STACK_CHECK</td>
<td>47</td>
</tr>
<tr>
<td>7.8.2.48 CH_DBG_FILL_THREADS</td>
<td>CH_DBG_FILL_THREADS</td>
<td>47</td>
</tr>
<tr>
<td>7.8.2.49 CH_DBG_THREADS_PROFILING</td>
<td>CH_DBG_THREADS_PROFILING</td>
<td>48</td>
</tr>
<tr>
<td>7.8.2.50 CH_CFG_SYSTEM_EXTRA_FIELDS</td>
<td>CH_CFG_SYSTEM_EXTRA_FIELDS</td>
<td>48</td>
</tr>
<tr>
<td>7.8.2.51 CH_CFG_SYSTEM_INIT_HOOK</td>
<td>CH_CFG_SYSTEM_INIT_HOOK</td>
<td>48</td>
</tr>
<tr>
<td>7.8.2.52 CH_CFG_THREAD_EXTRA_FIELDS</td>
<td>CH_CFG_THREAD_EXTRA_FIELDS</td>
<td>48</td>
</tr>
<tr>
<td>7.8.2.53 CH_CFG_THREAD_INIT_HOOK</td>
<td>CH_CFG_THREAD_INIT_HOOK</td>
<td>49</td>
</tr>
</tbody>
</table>
7.13.3.2 _idle_thread() .......................... 67
7.13.3.3 chSysInit() .............................. 67
7.13.3.4 chSysHalt() ............................... 69
7.13.3.5 chSysIntegrityCheckI() .................. 69
7.13.3.6 chSysTimerHandlerI() ..................... 70
7.13.3.7 chSysGetStatusAndLockX() ................. 71
7.13.3.8 chSysRestoreStatusX() ..................... 72
7.13.3.9 chSysIsCounterWithinX() .................. 73
7.13.3.10 chSysPolledDelayX() ....................... 74
7.13.3.11 chSysDisable() .......................... 74
7.13.3.12 chSysSuspend() .......................... 75
7.13.3.13 chSysEnable() ........................... 76
7.13.3.14 chSysLock() ............................. 76
7.13.3.15 chSysUnlock() ........................... 77
7.13.3.16 chSysLockFromISR() ...................... 77
7.13.3.17 chSysUnlockFromISR() ..................... 78
7.13.3.18 chSysUnconditionalLock() ................. 78
7.13.3.19 chSysUnconditionalUnlock() ............... 79
7.13.3.20 chSysGetIdleThreadX() .................... 80

7.14 Lists and Queues ............................ 81
7.14.1 Detailed Description ........................ 81
7.14.2 Macro Definition Documentation .............. 82
  7.14.2.1 _CH_QUEUE_DATA ....................... 82
  7.14.2.2 CH_QUEUE_DECL ........................ 82
7.14.3 Typedef Documentation ........................ 83
  7.14.3.1 ch_list_t ............................... 83
  7.14.3.2 ch_queue_t ............................... 83
  7.14.3.3 ch_priority_queue_t ...................... 83
7.14.4 Function Documentation ...................... 83
  7.14.4.1 ch_list_init() .......................... 83
  7.14.4.2 ch_list_isempty() ....................... 84
  7.14.4.3 ch_list_notempty() ....................... 84
  7.14.4.4 ch_list_push() .......................... 84
  7.14.4.5 ch_list_pop() ........................... 85
  7.14.4.6 ch_queue_init() .......................... 85
  7.14.4.7 ch_queue_isempty() ....................... 86
  7.14.4.8 ch_queue_notempty() ....................... 86
  7.14.4.9 ch_queue_insert() ......................... 87
  7.14.4.10 ch_queue_fifo_remove() .................... 87
  7.14.4.11 ch_queue_lifo_remove() .................... 87
  7.14.4.12 ch_queue_dequeue() ....................... 88
  7.14.4.13 ch_queue_init() .......................... 88
7.14.4.14 ch_queue_remove_highest() .................................................. 89
7.14.4.15 ch_queue_insert_behind() .................................................. 89
7.14.4.16 ch_queue_insert_ahead() .................................................. 90
7.15 Scheduler .................................................................................. 91
  7.15.1 Detailed Description ............................................................. 91
  7.15.2 Macro Definition Documentation ......................................... 95
    7.15.2.1 MSG_OK ........................................................................ 95
    7.15.2.2 MSG_TIMEOUT ............................................................. 95
    7.15.2.3 MSG_RESET ................................................................. 95
    7.15.2.4 NOPRIO ..................................................................... 95
    7.15.2.5 IDLEPRIO .................................................................... 95
    7.15.2.6 LOWPRIO .................................................................... 96
    7.15.2.7 NORMALPRIO ............................................................... 96
    7.15.2.8 HIGHTPRIO ................................................................. 96
    7.15.2.9 CH_STATE_READY ......................................................... 96
    7.15.2.10 CH_STATE_CURRENT ................................................... 96
    7.15.2.11 CH_STATE_WTSTART .................................................. 96
    7.15.2.12 CH_STATE_SUSPENDED .............................................. 97
    7.15.2.13 CH_STATE_QUEUED .................................................... 97
    7.15.2.14 CH_STATE_WTSEM ..................................................... 97
    7.15.2.15 CH_STATE_WTMTX ..................................................... 97
    7.15.2.16 CH_STATE_WTCOND .................................................. 97
    7.15.2.17 CH_STATE_SLEEPING ............................................... 97
    7.15.2.18 CH_STATE_WTEXIT ................................................... 98
    7.15.2.19 CH_STATE_WTREVT ................................................... 98
    7.15.2.20 CH_STATE_WTANDEVT ............................................. 98
    7.15.2.21 CH_STATE_SNDFMQ .................................................... 98
    7.15.2.22 CH_STATE_SNDFMQ .................................................... 98
    7.15.2.23 CH_STATE_WTMQ ....................................................... 98
    7.15.2.24 CH_STATE_FINAL ...................................................... 99
    7.15.2.25 CH_STATE_NAMES ....................................................... 99
    7.15.2.26 CH_FLAG_MODE_MASK ................................................. 99
    7.15.2.27 CH_FLAG_MODE_STATIC ............................................. 99
    7.15.2.28 CH_FLAG_MODE_HEAP ............................................... 99
    7.15.2.29 CH_FLAG_MODE_MPOOL .......................................... 100
    7.15.2.30 CH_FLAG_TERMINATE ................................................. 100
    7.15.2.31 firstprio ................................................................. 100
    7.15.2.32 currp ..................................................................... 100
    7.15.2.33 __CH_STRINGIFY ....................................................... 100
  7.15.3 Typedef Documentation ....................................................... 101
    7.15.3.1 delta_list_t .............................................................. 101
    7.15.3.2 thread_t ................................................................. 101

ChibiOS/RT
## 7.15.3 Thread Structures

- **7.15.3.3 thread_reference_t** ................................. 101
- **7.15.3.4 threads_list_t** .................................. 101
- **7.15.3.5 threads_queue_t** .................................. 101
- **7.15.3.6 ready_list_t** ...................................... 102
- **7.15.3.7 vfunc_t** ............................................ 102
- **7.15.3.8 virtual_timer_t** ................................... 102
- **7.15.3.9 virtual_timers_list_t** ............................. 102
- **7.15.3.10 system_debug_t** .................................. 102
- **7.15.3.11 ch_system_t** ..................................... 102

## 7.15.4 Function Documentation

- **7.15.4.1 ch_sch_prio_insert()** .............................. 102
- **7.15.4.2 _scheduler_init()** ................................ 103
- **7.15.4.3 chSchReadyI()** .................................... 103
- **7.15.4.4 chSchReadyAheadI()** .............................. 104
- **7.15.4.5 chSchGoSleepS()** .................................. 105
- **7.15.4.6 chSchGoSleepTimeoutS()** ......................... 106
- **7.15.4.7 chSchWakeupS()** .................................. 107
- **7.15.4.8 chSchRescheduleS()** ............................... 108
- **7.15.4.9 chSchIsPreemptionRequired()** .................. 109
- **7.15.4.10 chSchDoRescheduleBehind()** .................... 109
- **7.15.4.11 chSchDoRescheduleAhead()** ...................... 110
- **7.15.4.12 chSchDoReschedule()** ............................. 110
- **7.15.4.13 chSchIsRescRequiredI()** ......................... 111
- **7.15.4.14 chSchCanYieldS()** ................................ 112
- **7.15.4.15 chSchDoYieldS()** ................................ 113
- **7.15.4.16 chSchPreemption()** ............................... 113

## 7.15.5 Variable Documentation

- **7.15.5.1 ch** .................................................. 114

## 7.16 Time and Intervals

### 7.16.1 Detailed Description

### 7.16.2 Macro Definition Documentation

- **7.16.2.1 TIME_IMMEDIATE** ................................ 116
- **7.16.2.2 TIME_INFINITE** ................................... 116
- **7.16.2.3 TIME_MAX_INTERVAL** ............................... 116
- **7.16.2.4 TIME_MAX_SYSTIME** ............................... 117
- **7.16.2.5 TIME_S2I** ......................................... 117
- **7.16.2.6 TIME_MS2I** ........................................ 117
- **7.16.2.7 TIME_US2I** ........................................ 118
- **7.16.2.8 TIME_I2S** ......................................... 119
- **7.16.2.9 TIME_I2MS** ........................................ 119
- **7.16.2.10 TIME_I2US** ....................................... 120

## 7.16.3 Typedef Documentation

---

ChibiOS/RT
7.16.3.1 systime_t .............................................. 121
7.16.3.2 sysinterval_t ....................................... 121
7.16.3.3 time_secs_t .......................................... 121
7.16.3.4 time_msecs_t ....................................... 121
7.16.3.5 time_usecs_t ......................................... 122
7.16.3.6 time_conv_t ......................................... 122

7.16.4 Function Documentation .................................. 122
7.16.4.1 chTimeS2I() ........................................ 122
7.16.4.2 chTimeMS2I() ....................................... 123
7.16.4.3 chTimeUS2I() ....................................... 123
7.16.4.4 chTimeI2S() .......................................... 124
7.16.4.5 chTimeI2MS() ......................................... 124
7.16.4.6 chTimeI2US() ......................................... 125
7.16.4.7 chTimeAddX() ........................................ 125
7.16.4.8 chTimeDiffX() ........................................ 126
7.16.4.9 chTimelnInRangeX() .................................. 126

7.17 Virtual Timers ................................................... 128
7.17.1 Detailed Description ..................................... 128
7.17.2 Function Documentation .................................. 129
7.17.2.1 vt_init() .............................................. 129
7.17.2.2 vt_is_empty() ......................................... 129
7.17.2.3 vt_is_last() ........................................... 130
7.17.2.4 vt_is_first() .......................................... 130
7.17.2.5 vt_is_timer() .......................................... 131
7.17.2.6 vt_insert_after() ..................................... 131
7.17.2.7 vt_insert_before() .................................. 131
7.17.2.8 vt_insert() ........................................... 132
7.17.2.9 vt_remove_first() ................................... 132
7.17.2.10 vt_dequeue() ........................................ 133
7.17.2.11 vt_set_alarm() ....................................... 133
7.17.2.12 vt_insert_first() .................................... 134
7.17.2.13 vt_enqueue() .......................................... 135
7.17.2.14 _vt_init() ............................................. 135
7.17.2.15 chVTDoSetI() ......................................... 136
7.17.2.16 chVTDoResetI() ....................................... 137
7.17.2.17 chVTGetRemainingIntervalI() ....................... 138
7.17.2.18 chVTDoTickI() ....................................... 139
7.17.2.19 chVTOBJECTInit() .................................... 140
7.17.2.20 chVTGetSystemTimeX() ............................... 141
7.17.2.21 chVTGetSystemTime() ................................ 141
7.17.2.22 chVTTimeElapsedSinceX() .......................... 142
7.17.2.23 chVTlsSystemTimeWithinX() ........................ 143

ChibiOS/RT
7.18.4.19 chThdYield() ........................................ 178
7.18.4.20 chThdSuspend() ..................................... 179
7.18.4.21 chThdSuspendTimeout() ............................ 180
7.18.4.22 chThdResume() ..................................... 181
7.18.4.23 chThdResumeS() .................................... 182
7.18.4.24 chThdResume() ..................................... 182
7.18.4.25 chThdEnqueueTimeoutS() .......................... 183
7.18.4.26 chThdDequeueNextI() .............................. 184
7.18.4.27 chThdDequeueAllI() ............................... 185
7.18.4.28 chThdGetSelfX() ................................... 186
7.18.4.29 chThdGetPriorityX() .............................. 186
7.18.4.30 chThdGetTicksX() .................................. 186
7.18.4.31 chThdGetWorkingAreaX() ......................... 187
7.18.4.32 chThdTerminatedX() ............................... 187
7.18.4.33 chThdShouldTerminateX() ......................... 188
7.18.4.34 chThdStartI() ..................................... 188
7.18.4.35 chThdSleepS() ..................................... 189
7.18.4.36 chThdQueueObjectInit() .......................... 190
7.18.4.37 chThdQueueIsEmptyI() ............................ 190
7.18.4.38 chThdDoDequeueNextI() .......................... 191

7.19 Time Measurement ........................................ 193
  7.19.1 Detailed Description ................................ 193
  7.19.2 Macro Definition Documentation .................... 193
    7.19.2.1 TM_CALIBRATION_LOOP .......................... 193
  7.19.3 Function Documentation ............................ 194
    7.19.3.1 _tm_init() ..................................... 194
    7.19.3.2 chTMObjectInit() ............................... 194
    7.19.3.3 chTMStartMeasurementX() ....................... 195
    7.19.3.4 chTMStopMeasurementX() ....................... 195
    7.19.3.5 chTMChainMeasurementToX() .................... 195

7.20 Synchronization ......................................... 197
  7.20.1 Detailed Description ................................ 197

7.21 Counting Semaphores ................................... 198
  7.21.1 Detailed Description ............................... 198
  7.21.2 Macro Definition Documentation .................... 199
    7.21.2.1 _SEMAPHORE_DATA .............................. 199
    7.21.2.2 SEMAPHORE_DECL ............................... 200
  7.21.3 Typedef Documentation ............................. 200
    7.21.3.1 semaphore_t .................................... 200
  7.21.4 Function Documentation ............................ 200
    7.21.4.1 chSemObjectInit() ............................. 200
    7.21.4.2 chSemResetWithMessage() ..................... 201
7.21.4.3 chSemResetWithMessageI() ........................................ 202
7.21.4.4 chSemWait() ......................................................... 203
7.21.4.5 chSemWaitI() ..................................................... 204
7.21.4.6 chSemWaitTimeout() .............................................. 205
7.21.4.7 chSemWaitTimeoutS() ............................................. 206
7.21.4.8 chSemSignal() ....................................................... 207
7.21.4.9 chSemSignalI() .................................................... 208
7.21.4.10 chSemAddCounterI() ............................................. 209
7.21.4.11 chSemSignalWait() .............................................. 210
7.21.4.12 chSemReset() ...................................................... 211
7.21.4.13 chSemResetI() .................................................... 212
7.21.4.14 chSemFastWaitI() ............................................... 213
7.21.4.15 chSemFastSignalI() .............................................. 214
7.21.4.16 chSemGetCounterI() ............................................. 214

7.22 Mutexes ................................................................. 216

7.22.1 Detailed Description .................................................. 216
7.22.2 Macro Definition Documentation .................................. 217
  7.22.2.1 _MUTEX_DATA ................................................... 218
  7.22.2.2 MUTEX_DECL .................................................. 219
7.22.3 Typedef Documentation ............................................ 219
  7.22.3.1 mutex_t .......................................................... 219
7.22.4 Function Documentation .......................................... 219
  7.22.4.1 chMtxObjectInit() ............................................. 219
  7.22.4.2 chMtxLock() .................................................... 220
  7.22.4.3 chMtxLockS() .................................................. 221
  7.22.4.4 chMtxTryLock() ............................................... 222
  7.22.4.5 chMtxTryLockS() .............................................. 223
  7.22.4.6 chMtxUnlock() ................................................. 224
  7.22.4.7 chMtxUnlockS() .............................................. 225
  7.22.4.8 chMtxUnlockAllS() ............................................ 226
  7.22.4.9 chMtxUnlockAll() ............................................ 227
  7.22.4.10 chMtxQueueNotEmptyS() .................................... 228
  7.22.4.11 chMtxGetOwnerI() ......................................... 229
  7.22.4.12 chMtxGetNextMutexX() .................................... 230

7.23 Condition Variables .................................................. 231

7.23.1 Detailed Description ............................................... 231
7.23.2 Macro Definition Documentation ................................ 232
  7.23.2.1 _CONDVAR_DATA ............................................. 232
  7.23.2.2 CONDVAR_DECL ................................................ 232
7.23.3 Typedef Documentation .......................................... 232
  7.23.3.1 condition_variable_t ....................................... 232
7.23.4 Function Documentation .......................................... 232
7.23.4.1 chCondObjectInit() ................................................................. 233
7.23.4.2 chCondSignal() ................................................................. 233
7.23.4.3 chCondSignalI() ................................................................. 234
7.23.4.4 chCondBroadcast() ............................................................ 235
7.23.4.5 chCondBroadcastI() ............................................................. 236
7.23.4.6 chCondWait() ................................................................. 237
7.23.4.7 chCondWaitS() ................................................................. 238
7.23.4.8 chCondWaitTimeout() ......................................................... 239
7.23.4.9 chCondWaitTimeoutS() ....................................................... 241

7.24 Event Flags ................................................................. 243

7.24.1 Detailed Description .......................................................... 243
7.24.2 Macro Definition Documentation ........................................... 245
  7.24.2.1 ALL_EVENTS ................................................................. 245
  7.24.2.2 EVENT_MASK ................................................................. 245
  7.24.2.3 _EVENTSOURCE_DATA ...................................................... 245
  7.24.2.4 EVENTSOURCE_DECL ....................................................... 246
7.24.3 Typedef Documentation ...................................................... 246
  7.24.3.1 event_source_t .............................................................. 246
  7.24.3.2 evhandler_t ................................................................. 246
7.24.4 Function Documentation .................................................... 246
  7.24.4.1 chEvtRegisterMaskWithFlags() ......................................... 247
  7.24.4.2 chEvtUnregister() .......................................................... 247
  7.24.4.3 chEvtGetAndClearEventsI() ........................................... 248
  7.24.4.4 chEvtGetAndClearEvents() ............................................ 249
  7.24.4.5 chEvtAddEvents() .......................................................... 250
  7.24.4.6 chEvtBroadcastFlagsI() ................................................ 250
  7.24.4.7 chEvtGetAndClearFlags() .............................................. 251
  7.24.4.8 chEvtSignal() .............................................................. 252
  7.24.4.9 chEvtSignalI() ............................................................... 253
  7.24.4.10 chEvtBroadcastFlags() .................................................. 254
  7.24.4.11 chEvtGetAndClearFlagsI() ............................................ 255
  7.24.4.12 chEvtDispatch() ............................................................ 256
  7.24.4.13 chEvtWaitOne() ............................................................ 256
  7.24.4.14 chEvtWaitAny() ............................................................ 257
  7.24.4.15 chEvtWaitAll() ............................................................. 258
  7.24.4.16 chEvtWaitOneTimeout() ................................................ 259
  7.24.4.17 chEvtWaitAnyTimeout() ................................................ 260
  7.24.4.18 chEvtWaitAllTimeout() ................................................ 261
  7.24.4.19 chEvtObjectInit() ........................................................ 262
  7.24.4.20 chEvtRegisterMask() .................................................... 263
  7.24.4.21 chEvtRegister() ............................................................ 263
  7.24.4.22 chEvtIsListeningI() ..................................................... 264
7.24.4.23 chEvtBroadcast() .................................................. 265
7.24.4.24 chEvtBroadcastI() ................................................. 265
7.24.4.25 chEvtAddEventsI() ............................................... 266
7.24.4.26 chEvtGetEventsX() ............................................... 266

7.25 Synchronous Messages ................................................. 267
7.25.1 Detailed Description ............................................... 267
7.25.2 Function Documentation .......................................... 268
  7.25.2.1 chMsgSend() ...................................................... 268
  7.25.2.2 chMsgWaitS() .................................................... 268
  7.25.2.3 chMsgWaitTimeoutS() ........................................... 269
  7.25.2.4 chMsgPollS() ................................................... 270
  7.25.2.5 chMsgRelease() ................................................ 271
  7.25.2.6 chMsgWait() ..................................................... 272
  7.25.2.7 chMsgWaitTimeout() ............................................ 273
  7.25.2.8 chMsgPoll() .................................................... 274
  7.25.2.9 chMsgIsPendingI() .............................................. 276
  7.25.2.10 chMsgGet() .................................................... 276
  7.25.2.11 chMsgReleaseS() .............................................. 277

7.26 Dynamic Threads ........................................................... 278
7.26.1 Detailed Description ............................................... 278
7.26.2 Function Documentation .......................................... 278
  7.26.2.1 chThdCreateFromHeap() ...................................... 278
  7.26.2.2 chThdCreateFromMemoryPool() ................................ 279

7.27 Registry ..................................................................... 281
7.27.1 Detailed Description ............................................... 281
7.27.2 Macro Definition Documentation .................................. 282
  7.27.2.1 REG_REMOVE ..................................................... 282
  7.27.2.2 REG_INSERT ..................................................... 282
7.27.3 Function Documentation .......................................... 283
  7.27.3.1 chRegFirstThread() ............................................. 283
  7.27.3.2 chRegNextThread() ............................................. 283
  7.27.3.3 chRegFindThreadByName() .................................... 284
  7.27.3.4 chRegFindThreadByPointer() .................................. 285
  7.27.3.5 chRegFindThreadByWorkingArea() ......................... 286
  7.27.3.6 chRegSetThreadName() ....................................... 287
  7.27.3.7 chRegGetThreadNameX() .................................... 287
  7.27.3.8 chRegSetThreadNameX() .................................... 288

7.28 Debug ........................................................................ 289
7.28.1 Detailed Description ............................................... 289

7.29 Checks and Assertions .................................................... 290
7.29.1 Detailed Description ............................................... 290
7.29.2 Macro Definition Documentation .................................. 291
7.29.2.1 CH_DBG_STACK_FILL_VALUE ........................................... 291
7.29.2.2 chDbgCheck ............................................................... 292
7.29.2.3 chDbgAssert ............................................................... 292
7.29.3 Function Documentation .................................................... 293
  7.29.3.1 _dbg_check_disable() .................................................. 293
  7.29.3.2 _dbg_check_suspend() .................................................. 293
  7.29.3.3 _dbg_check_enable() .................................................... 294
  7.29.3.4 _dbg_check_lock() ...................................................... 294
  7.29.3.5 _dbg_check_unlock() .................................................... 295
  7.29.3.6 _dbg_check_lock_from_isr() ........................................... 295
  7.29.3.7 _dbg_check_unlock_from_isr() ....................................... 296
  7.29.3.8 _dbg_check_enter_isr() ............................................... 296
  7.29.3.9 _dbg_check_leave_isr() ................................................. 297
  7.29.3.10 chDbgCheckClassI() .................................................. 297
  7.29.3.11 chDbgCheckClassS() .................................................. 298
7.30 Tracing ................................................................................. 299
  7.30.1 Detailed Description ........................................................ 299
  7.30.2 Macro Definition Documentation ......................................... 300
    7.30.2.1 CH_DBG_TRACE_MASK .................................................. 300
    7.30.2.2 CH_DBG_TRACE_BUFFER_SIZE ....................................... 300
  7.30.3 Function Documentation .................................................... 301
    7.30.3.1 trace_next() ............................................................... 301
    7.30.3.2 _trace_init() ............................................................... 301
    7.30.3.3 _trace_switch() ............................................................ 301
    7.30.3.4 _trace_isr_enter() ...................................................... 302
    7.30.3.5 _trace_isr_leave() ...................................................... 302
    7.30.3.6 _trace_halt() ............................................................... 302
    7.30.3.7 chDbgWriteTraceI() ...................................................... 303
    7.30.3.8 chDbgWriteTrace() ........................................................ 303
    7.30.3.9 chDbgSuspendTraceI() .................................................. 304
    7.30.3.10 chDbgSuspendTrace() .................................................. 305
    7.30.3.11 chDbgResumeTraceI() .................................................. 306
    7.30.3.12 chDbgResumeTrace() ................................................... 306
7.31 Statistics .............................................................................. 308
  7.31.1 Detailed Description ........................................................ 308
  7.31.2 Function Documentation .................................................... 308
    7.31.2.1 _stats_init() ............................................................... 308
    7.31.2.2 _stats_increase_irq() .................................................... 309
    7.31.2.3 _stats_ctxswc() ............................................................. 309
    7.31.2.4 _stats_start_measure_crit_thd() .................................... 309
    7.31.2.5 _stats_stop_measure_crit_thd() ..................................... 310
    7.31.2.6 _stats_start_measure_crit_isr() .................................... 310
7.38.3.2 __ch_delegate_fn1() .......................................................... 356
7.38.3.3 __ch_delegate_fn2() .......................................................... 356
7.38.3.4 __ch_delegate_fn3() .......................................................... 356
7.38.3.5 __ch_delegate_fn4() .......................................................... 358
7.38.3.6 chDelegateCallVeneer() ...................................................... 358
7.38.3.7 chDelegateDispatch() ....................................................... 359
7.38.3.8 chDelegateDispatchTimeout() ............................................. 360
7.38.3.9 chDelegateCallDirect0() .................................................... 361
7.38.3.10 chDelegateCallDirect1() ................................................. 361
7.38.3.11 chDelegateCallDirect2() .................................................. 362
7.38.3.12 chDelegateCallDirect3() .................................................. 363
7.38.3.13 chDelegateCallDirect4() .................................................. 364

7.39 Jobs Queues ................................................................. 366
  7.39.1 Detailed Description ...................................................... 366
  7.39.2 Macro Definition Documentation ..................................... 367
    7.39.2.1 MSG_JOB_NULL .......................................................... 367
  7.39.3 Typedef Documentation .................................................. 367
    7.39.3.1 jobs_queue_t ............................................................. 367
    7.39.3.2 job_function_t .......................................................... 367
    7.39.3.3 job_descriptor_t ........................................................ 367
  7.39.4 Function Documentation ................................................ 367
    7.39.4.1 chJobObjectInit() ..................................................... 367
    7.39.4.2 chJobGet() ............................................................... 368
    7.39.4.3 chJobGetI() ................................................................ 369
    7.39.4.4 chJobGetTimeoutS() .................................................... 370
    7.39.4.5 chJobGetTimeout() ..................................................... 371
    7.39.4.6 chJobPostI() ............................................................... 372
    7.39.4.7 chJobPostS() ............................................................... 373
    7.39.4.8 chJobPost() ............................................................... 374
    7.39.4.9 chJobPostAheadI() ...................................................... 375
    7.39.4.10 chJobPostAheadS() ..................................................... 376
    7.39.4.11 chJobPostAhead() ...................................................... 377
    7.39.4.12 chJobDispatch() ......................................................... 378
    7.39.4.13 chJobDispatchTimeout() ............................................... 379

7.40 Memory Management ....................................................... 381
  7.40.1 Detailed Description ...................................................... 381

7.41 Core Memory Manager ..................................................... 382
  7.41.1 Detailed Description ...................................................... 382
  7.41.2 Macro Definition Documentation ..................................... 383
    7.41.2.1 CH_CFG_MEMCORE_SIZE ............................................. 383
    7.41.2.2 chCoreAllocAlignedWithOffsetI ..................................... 384
    7.41.2.3 chCoreAllocAlignedWithOffset ..................................... 384
7.41.3 Typedef Documentation
7.41.3.1 memgetfunc_t
7.41.3.2 memgetfunc2_t

7.41.4 Function Documentation
7.41.4.1 _core_init()
7.41.4.2 chCoreAllocFromBaseI()
7.41.4.3 chCoreAllocFromTopI()
7.41.4.4 chCoreAllocFromBase()
7.41.4.5 chCoreAllocFromTop()
7.41.4.6 chCoreGetStatusX()
7.41.4.7 chCoreAllocAlignedI()
7.41.4.8 chCoreAllocAligned()
7.41.4.9 chCoreAlloc()
7.41.4.10 chCoreAlloc()

7.41.5 Variable Documentation
7.41.5.1 ch_memcore

7.42 Memory Heaps
7.42.1 Detailed Description
7.42.2 Macro Definition Documentation
7.42.2.1 CH_HEAP_ALIGNMENT
7.42.2.2 CH_HEAP_AREA
7.42.3 Typedef Documentation
7.42.3.1 memory_heap_t
7.42.3.2 heap_header_t

7.42.4 Function Documentation
7.42.4.1 _heap_init()
7.42.4.2 chHeapObjectInit()
7.42.4.3 chHeapAllocAligned()
7.42.4.4 chHeapFree()
7.42.4.5 chHeapStatus()
7.42.4.6 chHeapAlloc()
7.42.4.7 chHeapGetSize()

7.42.5 Variable Documentation
7.42.5.1 default_heap

7.43 Memory Pools
7.43.1 Detailed Description
7.43.2 Macro Definition Documentation
7.43.2.1 _MEMORYPOOL_DATA
7.43.2.2 MEMORYPOOL_DECL
7.43.2.3 GUARDEDMEMORYPOOL_DATA
7.43.2.4 GUARDEDMEMORYPOOL_DECL

7.43.3 Function Documentation
7.47.3.4 dyn_buffer_t .................................................. 458
7.47.3.5 dyn_semaphore_t .............................................. 459
7.47.3.6 dyn_mailbox_t .................................................. 459
7.47.3.7 dyn_objects_fifo_t ............................................. 459
7.47.3.8 dyn_pipe_t ...................................................... 459
7.47.3.9 objects_factory_t .............................................. 459
7.47.4 Function Documentation ........................................ 459
  7.47.4.1 _factory_init() .............................................. 460
  7.47.4.2 chFactoryRegisterObject() ................................. 460
  7.47.4.3 chFactoryFindObject() ....................................... 461
  7.47.4.4 chFactoryFindObjectByPointer() ........................... 461
  7.47.4.5 chFactoryReleaseObject() ................................... 462
  7.47.4.6 chFactoryCreateBuffer() .................................... 462
  7.47.4.7 chFactoryFindBuffer() ...................................... 463
  7.47.4.8 chFactoryReleaseBuffer() ................................... 464
  7.47.4.9 chFactoryCreateSemaphore() ................................ 464
  7.47.4.10 chFactoryFindSemaphore() .................................. 465
  7.47.4.11 chFactoryReleaseSemaphore() ............................... 465
  7.47.4.12 chFactoryCreateMailbox() .................................. 466
  7.47.4.13 chFactoryFindMailbox() .................................... 466
  7.47.4.14 chFactoryReleaseMailbox() ................................ 467
  7.47.4.15 chFactoryCreateObjectsFIFO() ............................. 467
  7.47.4.16 chFactoryFindObjectsFIFO() ............................... 468
  7.47.4.17 chFactoryReleaseObjectsFIFO() ............................ 469
  7.47.4.18 chFactoryCreatePipe() ..................................... 469
  7.47.4.19 chFactoryFindPipe() ....................................... 470
  7.47.4.20 chFactoryReleasePipe() .................................... 470
  7.47.4.21 chFactoryDuplicateReference() ............................ 472
  7.47.4.22 chFactoryGetObject() ...................................... 472
  7.47.4.23 chFactoryGetBufferSize() .................................. 473
  7.47.4.24 chFactoryGetBuffer() ...................................... 473
  7.47.4.25 chFactoryGetSemaphore() ................................... 474
  7.47.4.26 chFactoryGetMailbox() .................................... 474
  7.47.4.27 chFactoryGetObjectsFIFO() ................................. 475
  7.47.4.28 chFactoryGetPipe() ....................................... 475
7.47.5 Variable Documentation ......................................... 476
  7.47.5.1 ch_factory .................................................... 476
8 Data Structure Documentation ......................................... 477
  8.1 ch_binary_semaphore Struct Reference ............................. 477
    8.1.1 Detailed Description .......................................... 478
  8.2 ch_delta_list Struct Reference ................................... 478
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>8.2.1</td>
<td>Detailed Description</td>
</tr>
<tr>
<td>8.2.2</td>
<td>Field Documentation</td>
</tr>
<tr>
<td>8.2.2.1</td>
<td>next</td>
</tr>
<tr>
<td>8.2.2.2</td>
<td>prev</td>
</tr>
<tr>
<td>8.2.2.3</td>
<td>delta</td>
</tr>
<tr>
<td>8.3</td>
<td>ch_dyn_element Struct Reference</td>
</tr>
<tr>
<td>8.3.1</td>
<td>Detailed Description</td>
</tr>
<tr>
<td>8.3.2</td>
<td>Field Documentation</td>
</tr>
<tr>
<td>8.3.2.1</td>
<td>next</td>
</tr>
<tr>
<td>8.3.2.2</td>
<td>refs</td>
</tr>
<tr>
<td>8.4</td>
<td>ch_dyn_list Struct Reference</td>
</tr>
<tr>
<td>8.4.1</td>
<td>Detailed Description</td>
</tr>
<tr>
<td>8.5</td>
<td>ch_dyn_mailbox Struct Reference</td>
</tr>
<tr>
<td>8.5.1</td>
<td>Detailed Description</td>
</tr>
<tr>
<td>8.5.2</td>
<td>Field Documentation</td>
</tr>
<tr>
<td>8.5.2.1</td>
<td>element</td>
</tr>
<tr>
<td>8.5.2.2</td>
<td>mbx</td>
</tr>
<tr>
<td>8.6</td>
<td>ch_dyn_object Struct Reference</td>
</tr>
<tr>
<td>8.6.1</td>
<td>Detailed Description</td>
</tr>
<tr>
<td>8.6.2</td>
<td>Field Documentation</td>
</tr>
<tr>
<td>8.6.2.1</td>
<td>element</td>
</tr>
<tr>
<td>8.7</td>
<td>ch_dyn_objects_fifo Struct Reference</td>
</tr>
<tr>
<td>8.7.1</td>
<td>Detailed Description</td>
</tr>
<tr>
<td>8.7.2</td>
<td>Field Documentation</td>
</tr>
<tr>
<td>8.7.2.1</td>
<td>element</td>
</tr>
<tr>
<td>8.7.2.2</td>
<td>fifo</td>
</tr>
<tr>
<td>8.8</td>
<td>ch_dyn_pipe Struct Reference</td>
</tr>
<tr>
<td>8.8.1</td>
<td>Detailed Description</td>
</tr>
<tr>
<td>8.8.2</td>
<td>Field Documentation</td>
</tr>
<tr>
<td>8.8.2.1</td>
<td>element</td>
</tr>
<tr>
<td>8.8.2.2</td>
<td>pipe</td>
</tr>
<tr>
<td>8.9</td>
<td>ch_dyn_semaphore Struct Reference</td>
</tr>
<tr>
<td>8.9.1</td>
<td>Detailed Description</td>
</tr>
<tr>
<td>8.9.2</td>
<td>Field Documentation</td>
</tr>
<tr>
<td>8.9.2.1</td>
<td>element</td>
</tr>
<tr>
<td>8.9.2.2</td>
<td>sem</td>
</tr>
<tr>
<td>8.10</td>
<td>ch_job_descriptor Struct Reference</td>
</tr>
<tr>
<td>8.10.1</td>
<td>Detailed Description</td>
</tr>
<tr>
<td>8.10.2</td>
<td>Field Documentation</td>
</tr>
<tr>
<td>8.10.2.1</td>
<td>jobfunc</td>
</tr>
<tr>
<td>8.10.2.2</td>
<td>jobarg</td>
</tr>
<tr>
<td>8.11</td>
<td>ch_jobs_queue Struct Reference</td>
</tr>
<tr>
<td>Section</td>
<td>Page</td>
</tr>
<tr>
<td>----------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>8.23 ch_semaphore Struct Reference</td>
<td>511</td>
</tr>
<tr>
<td>8.23.1 Detailed Description</td>
<td>512</td>
</tr>
<tr>
<td>8.23.2 Field Documentation</td>
<td>512</td>
</tr>
<tr>
<td>8.23.2.1 queue</td>
<td>512</td>
</tr>
<tr>
<td>8.23.2.2 cnt</td>
<td>513</td>
</tr>
<tr>
<td>8.24 ch_system Struct Reference</td>
<td>513</td>
</tr>
<tr>
<td>8.24.1 Detailed Description</td>
<td>514</td>
</tr>
<tr>
<td>8.24.2 Field Documentation</td>
<td>514</td>
</tr>
<tr>
<td>8.24.2.1 rlist</td>
<td>514</td>
</tr>
<tr>
<td>8.24.2.2 vtlist</td>
<td>514</td>
</tr>
<tr>
<td>8.24.2.3 dbg</td>
<td>514</td>
</tr>
<tr>
<td>8.24.2.4 mainthread</td>
<td>514</td>
</tr>
<tr>
<td>8.24.2.5 tm</td>
<td>515</td>
</tr>
<tr>
<td>8.24.2.6 kernel_stats</td>
<td>515</td>
</tr>
<tr>
<td>8.25 ch_system_debug Struct Reference</td>
<td>515</td>
</tr>
<tr>
<td>8.25.1 Detailed Description</td>
<td>517</td>
</tr>
<tr>
<td>8.25.2 Field Documentation</td>
<td>517</td>
</tr>
<tr>
<td>8.25.2.1 panic_msg</td>
<td>517</td>
</tr>
<tr>
<td>8.25.2.2 isr_cnt</td>
<td>517</td>
</tr>
<tr>
<td>8.25.2.3 lock_cnt</td>
<td>517</td>
</tr>
<tr>
<td>8.25.2.4 trace_buffer</td>
<td>517</td>
</tr>
<tr>
<td>8.26 ch_thread Struct Reference</td>
<td>518</td>
</tr>
<tr>
<td>8.26.1 Detailed Description</td>
<td>520</td>
</tr>
<tr>
<td>8.26.2 Field Documentation</td>
<td>520</td>
</tr>
<tr>
<td>8.26.2.1 list</td>
<td>520</td>
</tr>
<tr>
<td>8.26.2.2 queue</td>
<td>520</td>
</tr>
<tr>
<td>8.26.2.3 pqueue</td>
<td>520</td>
</tr>
<tr>
<td>8.26.2.4 ctx</td>
<td>520</td>
</tr>
<tr>
<td>8.26.2.5 newer</td>
<td>521</td>
</tr>
<tr>
<td>8.26.2.6 older</td>
<td>521</td>
</tr>
<tr>
<td>8.26.2.7 name</td>
<td>521</td>
</tr>
<tr>
<td>8.26.2.8 wabase</td>
<td>521</td>
</tr>
<tr>
<td>8.26.2.9 state</td>
<td>521</td>
</tr>
<tr>
<td>8.26.2.10 flags</td>
<td>521</td>
</tr>
<tr>
<td>8.26.2.11 refs</td>
<td>522</td>
</tr>
<tr>
<td>8.26.2.12 ticks</td>
<td>522</td>
</tr>
<tr>
<td>8.26.2.13 time</td>
<td>522</td>
</tr>
<tr>
<td>8.26.2.14 rdymsg</td>
<td>522</td>
</tr>
<tr>
<td>8.26.2.15 exitcode</td>
<td>523</td>
</tr>
<tr>
<td>8.26.2.16 wtobip</td>
<td>523</td>
</tr>
<tr>
<td>8.26.2.17 wttrp</td>
<td>523</td>
</tr>
<tr>
<td>8.26.2.18 sentmsg</td>
<td>523</td>
</tr>
</tbody>
</table>
8.26.2.19 wtsemp ............................................. 524
8.26.2.20 wtmxp ............................................. 524
8.26.2.21 ewmask ........................................... 524
8.26.2.22 u .................................................. 524
8.26.2.23 waiting .......................................... 525
8.26.2.24 msgqueue ......................................... 525
8.26.2.25 epending ......................................... 525
8.26.2.26 mtxlist .......................................... 525
8.26.2.27 realprio ......................................... 525
8.26.2.28 mpool .......................................... 525
8.26.2.29 stats ............................................ 526

8.27 ch_threads_queue Struct Reference ............................................. 526
8.27.1 Detailed Description ............................................. 526
8.27.2 Field Documentation ............................................. 526
   8.27.2.1 queue ........................................... 527

8.28 ch_trace_buffer_t Struct Reference ............................................. 527
8.28.1 Detailed Description ............................................. 529
8.28.2 Field Documentation ............................................. 529
   8.28.2.1 suspended ....................................... 529
   8.28.2.2 size ............................................. 529
   8.28.2.3 ptr ............................................. 529
   8.28.2.4 buffer ........................................... 529

8.29 ch_trace_event_t Struct Reference ............................................. 530
8.29.1 Detailed Description ............................................. 531
8.29.2 Field Documentation ............................................. 531
   8.29.2.1 type ............................................. 531
   8.29.2.2 state ............................................ 532
   8.29.2.3 rtstamp ......................................... 532
   8.29.2.4 time ............................................ 532
   8.29.2.5 ntp ............................................. 532
   8.29.2.6 wtobjp .......................................... 532
   8.29.2.7 sw .............................................. 532
   8.29.2.8 name ............................................ 533
   8.29.2.9 isr ............................................. 533
   8.29.2.10 reason ......................................... 533
   8.29.2.11 halt ............................................ 533
   8.29.2.12 up1 ............................................ 533
   8.29.2.13 up2 ............................................ 533
   8.29.2.14 user ............................................ 534

8.30 ch_virtual_timer Struct Reference ............................................. 534
8.30.1 Detailed Description ............................................. 534
8.30.2 Field Documentation ............................................. 535
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>9.17.1</td>
<td>Detailed Description</td>
<td>586</td>
</tr>
<tr>
<td>9.18</td>
<td>chfactory.h File Reference</td>
<td>586</td>
</tr>
<tr>
<td>9.18.1</td>
<td>Detailed Description</td>
<td>589</td>
</tr>
<tr>
<td>9.19</td>
<td>chjobs.h File Reference</td>
<td>589</td>
</tr>
<tr>
<td>9.19.1</td>
<td>Detailed Description</td>
<td>590</td>
</tr>
<tr>
<td>9.20</td>
<td>chlib.h File Reference</td>
<td>590</td>
</tr>
<tr>
<td>9.20.1</td>
<td>Detailed Description</td>
<td>591</td>
</tr>
<tr>
<td>9.21</td>
<td>chlicense.h File Reference</td>
<td>591</td>
</tr>
<tr>
<td>9.21.1</td>
<td>Detailed Description</td>
<td>592</td>
</tr>
<tr>
<td>9.22</td>
<td>chlists.h File Reference</td>
<td>592</td>
</tr>
<tr>
<td>9.22.1</td>
<td>Detailed Description</td>
<td>594</td>
</tr>
<tr>
<td>9.23</td>
<td>chmboxes.c File Reference</td>
<td>594</td>
</tr>
<tr>
<td>9.23.1</td>
<td>Detailed Description</td>
<td>594</td>
</tr>
<tr>
<td>9.24</td>
<td>chmboxes.h File Reference</td>
<td>594</td>
</tr>
<tr>
<td>9.24.1</td>
<td>Detailed Description</td>
<td>596</td>
</tr>
<tr>
<td>9.25</td>
<td>chmemcore.c File Reference</td>
<td>596</td>
</tr>
<tr>
<td>9.25.1</td>
<td>Detailed Description</td>
<td>596</td>
</tr>
<tr>
<td>9.26</td>
<td>chmemcore.h File Reference</td>
<td>596</td>
</tr>
<tr>
<td>9.26.1</td>
<td>Detailed Description</td>
<td>597</td>
</tr>
<tr>
<td>9.27</td>
<td>chmemheaps.c File Reference</td>
<td>597</td>
</tr>
<tr>
<td>9.27.1</td>
<td>Detailed Description</td>
<td>598</td>
</tr>
<tr>
<td>9.28</td>
<td>chmemheaps.h File Reference</td>
<td>598</td>
</tr>
<tr>
<td>9.28.1</td>
<td>Detailed Description</td>
<td>599</td>
</tr>
<tr>
<td>9.29</td>
<td>chmempools.c File Reference</td>
<td>599</td>
</tr>
<tr>
<td>9.29.1</td>
<td>Detailed Description</td>
<td>600</td>
</tr>
<tr>
<td>9.30</td>
<td>chmempools.h File Reference</td>
<td>600</td>
</tr>
<tr>
<td>9.30.1</td>
<td>Detailed Description</td>
<td>601</td>
</tr>
<tr>
<td>9.31</td>
<td>chmsg.c File Reference</td>
<td>601</td>
</tr>
<tr>
<td>9.31.1</td>
<td>Detailed Description</td>
<td>602</td>
</tr>
<tr>
<td>9.32</td>
<td>chmsg.h File Reference</td>
<td>602</td>
</tr>
<tr>
<td>9.32.1</td>
<td>Detailed Description</td>
<td>602</td>
</tr>
<tr>
<td>9.33</td>
<td>chmtx.c File Reference</td>
<td>602</td>
</tr>
<tr>
<td>9.33.1</td>
<td>Detailed Description</td>
<td>603</td>
</tr>
<tr>
<td>9.34</td>
<td>chmtx.h File Reference</td>
<td>603</td>
</tr>
<tr>
<td>9.34.1</td>
<td>Detailed Description</td>
<td>604</td>
</tr>
<tr>
<td>9.35</td>
<td>chobjcaches.c File Reference</td>
<td>604</td>
</tr>
<tr>
<td>9.35.1</td>
<td>Detailed Description</td>
<td>605</td>
</tr>
<tr>
<td>9.36</td>
<td>chobjcaches.h File Reference</td>
<td>605</td>
</tr>
<tr>
<td>9.36.1</td>
<td>Detailed Description</td>
<td>607</td>
</tr>
<tr>
<td>9.37</td>
<td>chobjfifos.h File Reference</td>
<td>607</td>
</tr>
<tr>
<td>9.37.1</td>
<td>Detailed Description</td>
<td>608</td>
</tr>
<tr>
<td>9.38</td>
<td>chpipes.c File Reference</td>
<td>608</td>
</tr>
<tr>
<td>Section</td>
<td></td>
<td></td>
</tr>
<tr>
<td>---------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9.38.1 Detailed Description</td>
<td>609</td>
<td></td>
</tr>
<tr>
<td>9.39 chpipes.h File Reference</td>
<td>609</td>
<td></td>
</tr>
<tr>
<td>9.39.1 Detailed Description</td>
<td>610</td>
<td></td>
</tr>
<tr>
<td>9.40 chregistry.c File Reference</td>
<td>610</td>
<td></td>
</tr>
<tr>
<td>9.40.1 Detailed Description</td>
<td>610</td>
<td></td>
</tr>
<tr>
<td>9.41 chregistry.h File Reference</td>
<td>611</td>
<td></td>
</tr>
<tr>
<td>9.41.1 Detailed Description</td>
<td>611</td>
<td></td>
</tr>
<tr>
<td>9.42 chrestrictions.h File Reference</td>
<td>611</td>
<td></td>
</tr>
<tr>
<td>9.42.1 Detailed Description</td>
<td>612</td>
<td></td>
</tr>
<tr>
<td>9.43 chschd.c File Reference</td>
<td>612</td>
<td></td>
</tr>
<tr>
<td>9.43.1 Detailed Description</td>
<td>612</td>
<td></td>
</tr>
<tr>
<td>9.44 chschd.h File Reference</td>
<td>613</td>
<td></td>
</tr>
<tr>
<td>9.44.1 Detailed Description</td>
<td>616</td>
<td></td>
</tr>
<tr>
<td>9.45 chsem.c File Reference</td>
<td>616</td>
<td></td>
</tr>
<tr>
<td>9.45.1 Detailed Description</td>
<td>616</td>
<td></td>
</tr>
<tr>
<td>9.46 chsem.h File Reference</td>
<td>616</td>
<td></td>
</tr>
<tr>
<td>9.46.1 Detailed Description</td>
<td>618</td>
<td></td>
</tr>
<tr>
<td>9.47 chstats.c File Reference</td>
<td>618</td>
<td></td>
</tr>
<tr>
<td>9.47.1 Detailed Description</td>
<td>618</td>
<td></td>
</tr>
<tr>
<td>9.48 chstats.h File Reference</td>
<td>618</td>
<td></td>
</tr>
<tr>
<td>9.48.1 Detailed Description</td>
<td>619</td>
<td></td>
</tr>
<tr>
<td>9.49 chsys.c File Reference</td>
<td>619</td>
<td></td>
</tr>
<tr>
<td>9.49.1 Detailed Description</td>
<td>620</td>
<td></td>
</tr>
<tr>
<td>9.50 chsys.h File Reference</td>
<td>620</td>
<td></td>
</tr>
<tr>
<td>9.50.1 Detailed Description</td>
<td>621</td>
<td></td>
</tr>
<tr>
<td>9.51 chsystypes.h File Reference</td>
<td>621</td>
<td></td>
</tr>
<tr>
<td>9.51.1 Detailed Description</td>
<td>622</td>
<td></td>
</tr>
<tr>
<td>9.52 chthreads.c File Reference</td>
<td>622</td>
<td></td>
</tr>
<tr>
<td>9.52.1 Detailed Description</td>
<td>623</td>
<td></td>
</tr>
<tr>
<td>9.53 chthreads.h File Reference</td>
<td>624</td>
<td></td>
</tr>
<tr>
<td>9.53.1 Detailed Description</td>
<td>626</td>
<td></td>
</tr>
<tr>
<td>9.54 chtime.h File Reference</td>
<td>626</td>
<td></td>
</tr>
<tr>
<td>9.54.1 Detailed Description</td>
<td>628</td>
<td></td>
</tr>
<tr>
<td>9.55 chtm.c File Reference</td>
<td>628</td>
<td></td>
</tr>
<tr>
<td>9.55.1 Detailed Description</td>
<td>628</td>
<td></td>
</tr>
<tr>
<td>9.56 chtm.h File Reference</td>
<td>628</td>
<td></td>
</tr>
<tr>
<td>9.56.1 Detailed Description</td>
<td>629</td>
<td></td>
</tr>
<tr>
<td>9.57 chtrace.c File Reference</td>
<td>629</td>
<td></td>
</tr>
<tr>
<td>9.57.1 Detailed Description</td>
<td>630</td>
<td></td>
</tr>
<tr>
<td>9.58 chtrace.h File Reference</td>
<td>630</td>
<td></td>
</tr>
<tr>
<td>9.58.1 Detailed Description</td>
<td>631</td>
<td></td>
</tr>
<tr>
<td>9.59 chversion.h File Reference</td>
<td>631</td>
<td></td>
</tr>
</tbody>
</table>
Chapter 1

ChibiOS/RT

1.1 Copyright

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

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

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

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

1.2 Introduction

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

1.3 Related Documents

- ChibiOS/RT General Architecture
Chapter 2

Kernel Concepts

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

- Naming Conventions
- API Name Suffixes
- Interrupt Classes
- System States
- Scheduling
- Thread States
- Priority Levels
- Thread Working Area

2.1 Naming Conventions

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

2.2 API Name Suffixes

The suffix can be one of the following:

- **None**, APIs without any suffix can be invoked only from the user code in the Normal state unless differently specified. See System States.
- "I", I-Class APIs are invokable only from the I-Locked or S-Locked states. See System States. This kind of functions never reschedule internally and are meant to be used from interrupt handlers. If used from thread level then an explicit reschedule operation could be necessary before leaving the containing critical zone.
- "S", S-Class APIs are invokable only from the S-Locked state. See System States. This kind of functions can reschedule internally but are not guaranteed to do so necessarily. For example a wait function on a semaphore can reschedule or not reschedule depending on the semaphore state.
- "X", X-Class APIs are invokable from any context.

Examples: chThdCreateStatic(), chSemSignalI(), chThdCreateStatic().
2.3 Interrupt Classes

In ChibiOS/RT there are three logical interrupt classes:

- **Regular Interrupts.** Maskable interrupt sources that cannot preempt (small parts of) the kernel code and are thus able to invoke operating system APIs from within their handlers. The interrupt handlers belonging to this class must be written following some rules. See the system APIs group and the web article *How to write interrupt handlers.*

- **Fast Interrupts.** Maskable interrupt sources with the ability to preempt the kernel code and thus have a lower latency and are less subject to jitter, see the web article *Response Time and Jitter.* Such sources are not supported on all the architectures. Fast interrupts are not allowed to invoke any operating system API from within their handlers. Fast interrupt sources may, however, pend a lower priority regular interrupt where access to the operating system is possible.

- **Non Maskable Interrupts.** Non maskable interrupt sources are totally out of the operating system control and have the lowest latency. Such sources are not supported on all the architectures.

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

2.4 System States

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

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

- **Normal.** All the interrupt sources are enabled and the system APIs are accessible, threads are running.

- **Suspended.** In this state the fast interrupt sources are enabled but the regular interrupt sources are not. In this state it is not possible to use any system API except `chSysDisable()` or `chSysEnable()` in order to change state.

- **Disabled.** When the system is in this state both the maskable regular and fast interrupt sources are disabled. In this state it is not possible to use any system API except `chSysSuspend()` or `chSysEnable()` in order to change state.

- **Sleep.** Architecture-dependent low power mode, the idle thread goes in this state and waits for interrupts, after servicing the interrupt the Normal state is restored and the scheduler has a chance to reschedule.

- **S-Locked.** Kernel locked and regular interrupt sources disabled. Fast interrupt sources are enabled. **S-Class** APIs are invokable in this state.

- **I-Locked.** Kernel locked and regular interrupt sources disabled. **I-Class** APIs are invokable from this state.

- **Serving Regular Interrupt.** No system APIs are accessible but it is possible to switch to the I-Locked state using `chSysLockFromIsr()` and then invoke any **I-Class** API. Interrupt handlers can be preemptable on some architectures thus is important to switch to I-Locked state before invoking system APIs.

- **Serving Fast Interrupt.** System APIs are not accessible.

- **Serving Non-Maskable Interrupt.** System APIs are not accessible.

- **Halted.** All interrupt sources are disabled and system stopped into an infinite loop. This state can be reached if the debug mode is activated and an error is detected or after explicitly invoking `chSysHalt()`.

ChibiOS/RT
Note that the above states are just **Logical States** that may have no real associated machine state on some architectures. The following diagram shows the possible transitions between the states:

```
Init
<table>
<thead>
<tr>
<th>chSysInit()</th>
</tr>
</thead>
<tbody>
<tr>
<td>Normal</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Suspended</td>
</tr>
<tr>
<td>Disabled</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>S-Locked</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Sleep</td>
</tr>
<tr>
<td>Idle Thread</td>
</tr>
<tr>
<td>Regular IRQ</td>
</tr>
<tr>
<td>Regular IRQ return</td>
</tr>
<tr>
<td>SRI</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>I-Locked</td>
</tr>
</tbody>
</table>

Note, the **SFI**, **Halted** and **SNMI** states were not shown because those are reachable from most states:

```
Any State except * |
Fast IRQ |
| Fast IRQ return |
SFI |
```
### 2.5 Scheduling

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

Note that the currently running thread is not in the ready list, the list only contains the threads ready to be executed but still actually waiting.

---

*Attention*

* except: Init, Halt, SNMI, Disabled.
2.6 Thread States

The image shows how threads can change their state in ChibiOS/RT.

2.7 Priority Levels

Priorities in ChibiOS/RT are a contiguous numerical range but the initial and final values are not enforced. The following table describes the various priority boundaries (from lowest to highest):

- **IDLEPRIO**, this is the lowest priority level and is reserved for the idle thread, no other threads should share this priority level. This is the lowest numerical value of the priorities space.
- **LOWPRIO**, the lowest priority level that can be assigned to an user thread.
- **NORMALPRIO**, this is the central priority level for user threads. It is advisable to assign priorities to threads as values relative to **NORMALPRIO**, as example **NORMALPRIO-1** or **NORMALPRIO+4**, this ensures the portability of code should the numerical range change in future implementations.
- **HIGHPRIO**, the highest priority level that can be assigned to an user thread.
- **ABSPRO**, absolute maximum software priority level, it can be higher than **HIGHPRIO** but the numerical values above **HIGHPRIO** up to **ABSPRO** (inclusive) are reserved. This is the highest numerical value of the priorities space.
2.8 Thread Working Area

Each thread has its own stack, a Thread structure and some preemption areas. All the structures are allocated into a "Thread Working Area", a thread private heap, usually statically declared in your code. Threads do not use any memory outside the allocated working area except when accessing static shared data.

Note that the preemption area is only present when the thread is not running (switched out), the context switching is done by pushing the registers on the stack of the switched-out thread and popping the registers of the switched-in thread from its stack. The preemption area can be divided in up to three structures:

- External Context.
- Interrupt Stack.
- Internal Context.

See the port documentation for details, the area may change on the various ports and some structures may not be present (or be zero-sized).
# Chapter 3

## Module Index

### 3.1 Modules

Here is a list of all modules:

<table>
<thead>
<tr>
<th>Module</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Release and Licensing</td>
<td>19</td>
</tr>
<tr>
<td>Release Information</td>
<td>20</td>
</tr>
<tr>
<td>Customer Information</td>
<td>22</td>
</tr>
<tr>
<td>License Settings</td>
<td>24</td>
</tr>
<tr>
<td>RT Kernel</td>
<td>26</td>
</tr>
<tr>
<td>Version Numbers and Identification</td>
<td>27</td>
</tr>
<tr>
<td>Configuration</td>
<td>31</td>
</tr>
<tr>
<td>Options</td>
<td>32</td>
</tr>
<tr>
<td>Checks</td>
<td>52</td>
</tr>
<tr>
<td>Restrictions</td>
<td>53</td>
</tr>
<tr>
<td>Base Kernel Services</td>
<td>54</td>
</tr>
<tr>
<td>Memory Alignment</td>
<td>55</td>
</tr>
<tr>
<td>System Management</td>
<td>58</td>
</tr>
<tr>
<td>Lists and Queues</td>
<td>81</td>
</tr>
<tr>
<td>Scheduler</td>
<td>91</td>
</tr>
<tr>
<td>Time and Intervals</td>
<td>115</td>
</tr>
<tr>
<td>Virtual Timers</td>
<td>128</td>
</tr>
<tr>
<td>Threads</td>
<td>153</td>
</tr>
<tr>
<td>Time Measurement</td>
<td>193</td>
</tr>
<tr>
<td>Synchronization</td>
<td>197</td>
</tr>
<tr>
<td>Counting Semaphores</td>
<td>198</td>
</tr>
<tr>
<td>Mutexes</td>
<td>216</td>
</tr>
<tr>
<td>Condition Variables</td>
<td>231</td>
</tr>
<tr>
<td>Event Flags</td>
<td>243</td>
</tr>
<tr>
<td>Synchronous Messages</td>
<td>267</td>
</tr>
<tr>
<td>Dynamic Threads</td>
<td>278</td>
</tr>
<tr>
<td>Registry</td>
<td>281</td>
</tr>
<tr>
<td>Debug</td>
<td>289</td>
</tr>
<tr>
<td>Checks and Assertions</td>
<td>290</td>
</tr>
<tr>
<td>Tracing</td>
<td>299</td>
</tr>
<tr>
<td>Statistics</td>
<td>308</td>
</tr>
<tr>
<td>OS Library</td>
<td>312</td>
</tr>
<tr>
<td>Version Numbers and Identification</td>
<td>313</td>
</tr>
<tr>
<td>Synchronization</td>
<td>316</td>
</tr>
</tbody>
</table>
Module Index

Binary Semaphores ................................................. 317
Mailboxes .......................................................... 328
Pipes ................................................................. 346
Delegate Threads ................................................... 354
Jobs Queues ......................................................... 366
Memory Management ............................................... 381
  Core Memory Manager ......................................... 382
  Memory Heaps .................................................... 393
  Memory Pools ..................................................... 400
Complex Services .................................................. 425
  Objects FIFOs ..................................................... 426
  Objects Caches ................................................... 443
  Dynamic Objects Factory ...................................... 453
Chapter 4

Hierarchical Index

4.1 Class Hierarchy

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

- ch_delta_list .......................................................... 478
- ch_dyn_element ......................................................... 480
- ch_dyn_list ............................................................ 481
- ch_dyn_mailbox ......................................................... 481
- ch_dyn_object ........................................................ 483
- ch_dyn_objects_fifo .................................................. 484
- ch_dyn_pipe ........................................................... 485
- ch_dyn_semaophore ................................................... 487
- ch_job_descriptor .................................................... 488
- ch_jobs_queue ......................................................... 489
- ch_list ................................................................. 491
- ch_mutex ............................................................... 492
- ch_objects_cache ...................................................... 494
- ch_objects_factory .................................................... 497
- ch_objects_fio ........................................................ 500
- ch_oc_hash_header ................................................... 501
- ch_oc_fru_header ..................................................... 503
- ch_oc_object .......................................................... 505
- ch_priority_queue ..................................................... 508
- ch_queue ............................................................... 509
- ch_registered_static_object ....................................... 510
- ch_semaphore .......................................................... 511
  - ch_binary_semaphore .............................................. 477
- ch_system ............................................................. 513
- ch_system_debug ....................................................... 515
- ch_thread ............................................................. 518
- ch_threads_queue ..................................................... 526
- ch_trace_buffer_t ..................................................... 527
- ch_trace_event_t ....................................................... 530
- ch_virtual_timer ..................................................... 534
- ch_virtual_timers_list .............................................. 535
- chdebug_t .............................................................. 537
- condition_variable .................................................. 541
- event_listener ......................................................... 542
- event_source .......................................................... 544
<table>
<thead>
<tr>
<th>Type</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>guarded_memory_pool_t</td>
<td>546</td>
</tr>
<tr>
<td>heap_header</td>
<td>547</td>
</tr>
<tr>
<td>kernel_stats_t</td>
<td>549</td>
</tr>
<tr>
<td>mailbox_t</td>
<td>551</td>
</tr>
<tr>
<td>memcore_t</td>
<td>554</td>
</tr>
<tr>
<td>memory_heap</td>
<td>555</td>
</tr>
<tr>
<td>memory_pool_t</td>
<td>557</td>
</tr>
<tr>
<td>pipe_t</td>
<td>558</td>
</tr>
<tr>
<td>pool_header</td>
<td>562</td>
</tr>
<tr>
<td>thread_descriptor_t</td>
<td>563</td>
</tr>
<tr>
<td>time_measurement_t</td>
<td>565</td>
</tr>
<tr>
<td>tm_calibration_t</td>
<td>567</td>
</tr>
</tbody>
</table>
# Chapter 5

## Data Structure Index

### 5.1 Data Structures

Here are the data structures with brief descriptions:

- **ch_binary_semaphore**: Binary semaphore type .................................................. 477
- **ch_delta_list**: Virtual Timer delta list element and header structure .................. 478
- **ch_dyn_element**: Type of a dynamic object list element .................................. 480
- **ch_dyn_list**: Type of a dynamic object list ....................................................... 481
- **ch_dyn_mailbox**: Type of a dynamic buffer object ............................................ 481
- **ch_dyn_object**: Type of a dynamic buffer object ............................................... 483
- **ch_dyn_objects_fifo**: Type of a dynamic buffer object ...................................... 484
- **ch_dyn_pipe**: Type of a dynamic pipe object .................................................... 485
- **ch_dyn_semaphore**: Type of a dynamic semaphore ............................................ 487
- **ch_job_descriptor**: Type of a job descriptor ..................................................... 488
- **ch_jobs_queue**: Type of a jobs queue ............................................................... 489
- **ch_list**: Structure representing a generic single link list header and element ...... 491
- **ch_mutex**: Mutex structure .............................................................................. 492
- **ch_objects_cache**: Structure representing a cache object .................................. 494
- **ch_objects_factory**: Type of the factory main object ........................................ 497
- **ch_objects_fifo**: Type of an objects FIFO ......................................................... 500
- **ch_oc_hash_header**: Structure representing an hash table element .................. 501
- **ch_oc_lru_header**: Structure representing an hash table element ..................... 503
<table>
<thead>
<tr>
<th>Data Structure Index</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ch_oc_object</strong></td>
</tr>
<tr>
<td>Structure representing a cached object ........................................... 505</td>
</tr>
<tr>
<td><strong>ch_priority_queue</strong></td>
</tr>
<tr>
<td>Structure representing a generic priority-ordered bidirectional linked list header and element . 508</td>
</tr>
<tr>
<td><strong>ch_queue</strong></td>
</tr>
<tr>
<td>Structure representing a generic bidirectional linked list header and element ............................... 509</td>
</tr>
<tr>
<td><strong>ch_registered_static_object</strong></td>
</tr>
<tr>
<td>Type of a registered object ............................................................ 510</td>
</tr>
<tr>
<td><strong>ch_semaphore</strong></td>
</tr>
<tr>
<td>Semaphore structure ................................................................. 511</td>
</tr>
<tr>
<td><strong>ch_system</strong></td>
</tr>
<tr>
<td>System data structure .............................................................. 513</td>
</tr>
<tr>
<td><strong>ch_system_debug</strong></td>
</tr>
<tr>
<td>System debug data structure ..................................................... 515</td>
</tr>
<tr>
<td><strong>ch_thread</strong></td>
</tr>
<tr>
<td>Structure representing a thread .................................................. 518</td>
</tr>
<tr>
<td><strong>ch_threads_queue</strong></td>
</tr>
<tr>
<td>Structure representing a threads queue ........................................ 526</td>
</tr>
<tr>
<td><strong>ch_trace_buffer_t</strong></td>
</tr>
<tr>
<td>Trace buffer header .............................................................. 527</td>
</tr>
<tr>
<td><strong>ch_trace_event_t</strong></td>
</tr>
<tr>
<td>Trace buffer record ........................................................... 530</td>
</tr>
<tr>
<td><strong>ch_virtual_timer</strong></td>
</tr>
<tr>
<td>Structure representing a Virtual Timer .......................................... 534</td>
</tr>
<tr>
<td><strong>ch_virtual_timers_list</strong></td>
</tr>
<tr>
<td>Structure representing a virtual timers list header ............................ 535</td>
</tr>
<tr>
<td><strong>chdebug_t</strong></td>
</tr>
<tr>
<td>ChibiOS/RT memory signature record ........................................... 537</td>
</tr>
<tr>
<td><strong>condition_variable</strong></td>
</tr>
<tr>
<td>Condition_variable_t structure .................................................... 541</td>
</tr>
<tr>
<td><strong>event_listener</strong></td>
</tr>
<tr>
<td>Event Listener structure .......................................................... 542</td>
</tr>
<tr>
<td><strong>event_source</strong></td>
</tr>
<tr>
<td>Event Source structure .......................................................... 544</td>
</tr>
<tr>
<td><strong>guarded_memory_pool_t</strong></td>
</tr>
<tr>
<td>Guarded memory pool descriptor .................................................. 546</td>
</tr>
<tr>
<td><strong>heap_header</strong></td>
</tr>
<tr>
<td>Memory heap block header ............................................................ 547</td>
</tr>
<tr>
<td><strong>kernel_stats_t</strong></td>
</tr>
<tr>
<td>Type of a kernel statistics structure ........................................... 549</td>
</tr>
<tr>
<td><strong>mailbox_t</strong></td>
</tr>
<tr>
<td>Structure representing a mailbox object ....................................... 551</td>
</tr>
<tr>
<td><strong>memcore_t</strong></td>
</tr>
<tr>
<td>Type of memory core object .......................................................... 554</td>
</tr>
<tr>
<td><strong>memory_heap</strong></td>
</tr>
<tr>
<td>Structure describing a memory heap ............................................ 555</td>
</tr>
<tr>
<td><strong>memory_pool_t</strong></td>
</tr>
<tr>
<td>Memory pool descriptor ............................................................ 557</td>
</tr>
<tr>
<td><strong>pipe_t</strong></td>
</tr>
<tr>
<td>Structure representing a pipe object .......................................... 558</td>
</tr>
<tr>
<td><strong>pool_header</strong></td>
</tr>
<tr>
<td>Memory pool free object header ................................................ 562</td>
</tr>
<tr>
<td><strong>thread_descriptor_t</strong></td>
</tr>
<tr>
<td>Type of a thread descriptor ................................................... 563</td>
</tr>
<tr>
<td><strong>time_measurement_t</strong></td>
</tr>
<tr>
<td>Type of a Time Measurement object ............................................ 565</td>
</tr>
<tr>
<td><strong>tm_calibration_t</strong></td>
</tr>
<tr>
<td>Type of a time measurement calibration data .................................... 567</td>
</tr>
</tbody>
</table>
6.1 File List

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

- **ch.h**: ChibiOS/RT main include file .................................................. 569
- **chalign.h**: Memory alignment macros and structures .................................. 570
- **chbsem.h**: Binary semaphores structures and macros .................................. 571
- **chchecks.h**: Configuration file checks header ........................................... 572
- **chcond.c**: Condition Variables code .................................................... 572
- **chcond.h**: Condition Variables macros and structures .................................. 573
- **chconf.h**: Configuration file template .................................................. 574
- **chcustomer.h**: Customer-related info .................................................... 577
- **chdebug.c**: Debug support code .............................................................. 578
- **chdebug.h**: Debug support macros and structures ....................................... 578
- **chdelegates.c**: Delegate threads code .................................................... 579
- **chdelegates.h**: Delegate threads macros and structures ............................... 580
- **chdynamic.c**: Dynamic threads code ...................................................... 581
- **chdynamic.h**: Dynamic threads macros and structures ................................. 581
- **chevents.c**: Events code ........................................................................ 582
- **chevents.h**: Events macros and structures .................................................. 583
- **chfactory.c**: ChibiOS objects factory and registry code ............................... 585
- **chfactory.h**: ChibiOS objects factory structures and macros ......................... 586
<table>
<thead>
<tr>
<th>File Name</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>chjobs.h</td>
<td>Jobs Queues structures and macros</td>
<td>589</td>
</tr>
<tr>
<td>chlib.h</td>
<td>ChibiOS/LIB main include file</td>
<td>590</td>
</tr>
<tr>
<td>chlicense.h</td>
<td>License Module macros and structures</td>
<td>591</td>
</tr>
<tr>
<td>chlists.h</td>
<td>Lists and Queues header</td>
<td>592</td>
</tr>
<tr>
<td>chmboxes.c</td>
<td>Mailboxes code</td>
<td>594</td>
</tr>
<tr>
<td>chmboxes.h</td>
<td>Mailboxes macros and structures</td>
<td>594</td>
</tr>
<tr>
<td>chmemcore.c</td>
<td>Core memory manager code</td>
<td>596</td>
</tr>
<tr>
<td>chmemcore.h</td>
<td>Core memory manager macros and structures</td>
<td>596</td>
</tr>
<tr>
<td>chmemheaps.c</td>
<td>Memory heaps code</td>
<td>597</td>
</tr>
<tr>
<td>chmemheaps.h</td>
<td>Memory heaps macros and structures</td>
<td>598</td>
</tr>
<tr>
<td>chmpools.c</td>
<td>Memory Pools code</td>
<td>599</td>
</tr>
<tr>
<td>chmpools.h</td>
<td>Memory Pools macros and structures</td>
<td>600</td>
</tr>
<tr>
<td>chmsg.c</td>
<td>Messages code</td>
<td>601</td>
</tr>
<tr>
<td>chmsg.h</td>
<td>Messages macros and structures</td>
<td>602</td>
</tr>
<tr>
<td>chmtx.c</td>
<td>Mutexes code</td>
<td>602</td>
</tr>
<tr>
<td>chmtx.h</td>
<td>Mutexes macros and structures</td>
<td>603</td>
</tr>
<tr>
<td>chobjcaches.c</td>
<td>Objects Caches code</td>
<td>604</td>
</tr>
<tr>
<td>chobjcaches.h</td>
<td>Objects Caches macros and structures</td>
<td>605</td>
</tr>
<tr>
<td>chobjfifos.h</td>
<td>Objects FIFO structures and macros</td>
<td>607</td>
</tr>
<tr>
<td>chpipes.c</td>
<td>Pipes code</td>
<td>608</td>
</tr>
<tr>
<td>chpipes.h</td>
<td>Pipes macros and structures</td>
<td>609</td>
</tr>
<tr>
<td>chregistry.c</td>
<td>Threads registry code</td>
<td>610</td>
</tr>
<tr>
<td>chregistry.h</td>
<td>Threads registry macros and structures</td>
<td>611</td>
</tr>
<tr>
<td>chrestrictions.h</td>
<td>Licensing restrictions header</td>
<td>611</td>
</tr>
<tr>
<td>chschd.c</td>
<td>Scheduler code</td>
<td>612</td>
</tr>
<tr>
<td>chschd.h</td>
<td>Scheduler macros and structures</td>
<td>613</td>
</tr>
<tr>
<td>chsem.c</td>
<td>Semaphores code</td>
<td>616</td>
</tr>
<tr>
<td>chsem.h</td>
<td>Semaphores macros and structures</td>
<td>616</td>
</tr>
<tr>
<td>chstats.c</td>
<td>Statistics module code</td>
<td>618</td>
</tr>
<tr>
<td>Header</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
<td>------</td>
</tr>
<tr>
<td>chstates.h</td>
<td>Statistics module macros and structures</td>
<td>618</td>
</tr>
<tr>
<td>chsys.c</td>
<td>System related code</td>
<td>619</td>
</tr>
<tr>
<td>chsys.h</td>
<td>System related macros and structures</td>
<td>620</td>
</tr>
<tr>
<td>chsys.h</td>
<td>System types header</td>
<td>621</td>
</tr>
<tr>
<td>chthreads.c</td>
<td>Threads code</td>
<td>622</td>
</tr>
<tr>
<td>chthread.h</td>
<td>Threads module macros and structures</td>
<td>624</td>
</tr>
<tr>
<td>chtime.h</td>
<td>Time and intervals macros and structures</td>
<td>626</td>
</tr>
<tr>
<td>chtm.h</td>
<td>Time Measurement module code</td>
<td>628</td>
</tr>
<tr>
<td>chtm.h</td>
<td>Time Measurement module macros and structures</td>
<td>628</td>
</tr>
<tr>
<td>chtrace.c</td>
<td>Tracer code</td>
<td>629</td>
</tr>
<tr>
<td>chtrace.h</td>
<td>Tracer macros and structures</td>
<td>630</td>
</tr>
<tr>
<td>chversion.h</td>
<td>Version Module macros and structures</td>
<td>631</td>
</tr>
<tr>
<td>chvt.c</td>
<td>Time and Virtual Timers module code</td>
<td>632</td>
</tr>
<tr>
<td>chvt.h</td>
<td>Time and Virtual Timers module macros and structures</td>
<td>633</td>
</tr>
</tbody>
</table>
Chapter 7

Module Documentation

7.1 Release and Licensing

7.1.1 Detailed Description

Modules

- Release Information
- Customer Information
- License Settings
7.2 Release Information

7.2.1 Detailed Description

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

Macros

- `#define _CHIBIOS_`
  
  ChibiOS product identification macro.

- `#define CH_VERSION_STABLE 1`
  
  Stable release flag.

ChibiOS version identification

- `#define CH_VERSION "20.3.4"`
  
  ChibiOS version string.

- `#define CH_VERSION_YEAR 20`
  
  ChibiOS version release year.

- `#define CH_VERSION_MONTH 3`
  
  ChibiOS version release month.

- `#define CH_VERSION_PATCH 4`
  
  ChibiOS version patch number.

- `#define CH_VERSION_NICKNAME "Conca"`
  
  ChibiOS version nickname.

7.2.2 Macro Definition Documentation

7.2.2.1 _CHIBIOS_

`#define _CHIBIOS_`

ChibiOS product identification macro.

7.2.2.2 CH_VERSION_STABLE

`#define CH_VERSION_STABLE 1`

Stable release flag.
7.2 Release Information

7.2.2.3  CH_VERSION

#define CH_VERSION "20.3.4"
ChibiOS version string.

7.2.2.4  CH_VERSION_YEAR

#define CH_VERSION_YEAR 20
ChibiOS version release year.

7.2.2.5  CH_VERSION_MONTH

#define CH_VERSION_MONTH 3
ChibiOS version release month.

7.2.2.6  CH_VERSION_PATCH

#define CH_VERSION_PATCH 4
ChibiOS version patch number.

7.2.2.7  CH_VERSION_NICKNAME

#define CH_VERSION_NICKNAME "Conca"
ChibiOS version nickname.
7.3 Customer Information

7.3.1 Detailed Description

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

Macros

- #define CH_CUSTOMER_ID_STRING "Santa, North Pole"
  Customer readable identifier.
- #define CH_CUSTOMER_ID_CODE "xxxx-yyyy"
  Customer code.
- #define CH_LICENSE CH_LICENSE_GPL
  Current license.

Licensed Products

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

7.3.2 Macro Definition Documentation

7.3.2.1 CH_CUSTOMER_ID_STRING

#define CH_CUSTOMER_ID_STRING "Santa, North Pole"

Customer readable identifier.
7.3.2.2 CH_CUSTOMER_ID_CODE

#define CH_CUSTOMER_ID_CODE "xxxx-yyyy"

Customer code.

7.3.2.3 CH_LICENSE

#define CH_LICENSE CH_LICENSE_GPL

Current license.

Note

This setting is reserved to the copyright owner.
Changing this setting invalidates the license.
The license statement in the source headers is valid, applicable and binding regardless this setting.
7.4 License Settings

7.4.1 Detailed Description

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

Macros

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

Allowed Features Levels

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

Deployment Options

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

Licensing Options

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

7.4.2 Macro Definition Documentation
7.4 License Settings

7.4.2.1 CH_LICENSE_TYPE_STRING

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

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

7.4.2.2 CH_LICENSE_ID_STRING

#define CH_LICENSE_ID_STRING "N/A"

Customer identification string.
This information is only available for registered commercial users.

7.4.2.3 CH_LICENSE_ID_CODE

#define CH_LICENSE_ID_CODE "N/A"

Customer code.
This information is only available for registered commercial users.

7.4.2.4 CH_LICENSE_MODIFIABLE_CODE

#define CH_LICENSE_MODIFIABLE_CODE TRUE

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

7.4.2.5 CH_LICENSE_FEATURES

#define CH_LICENSE_FEATURES CH_FEATURES_FULL

Code functionality restrictions.

7.4.2.6 CH_LICENSE_MAX_DEPLOY

#define CH_LICENSE_MAX_DEPLOY CH_DEPLOY_UNLIMITED

Code deploy restrictions.
This is the per-core deploy limit allowed under the current license scheme.
7.5 RT Kernel

7.5.1 Detailed Description

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

Modules

- Version Numbers and Identification
- Configuration
- Base Kernel Services
- Synchronization
- Dynamic Threads
- Registry
- Debug
7.6 Version Numbers and Identification

7.6.1 Detailed Description

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

Kernel related info.

Macros

- `#define _CHIBIOS_RT_`
  
  ChibiOS/RT identification macro.
- `#define CH_KERNEL_STABLE 1`
  
  Stable release flag.

ChibiOS/RT version identification

- `#define CH_KERNEL_VERSION "6.1.4"`
  
  Kernel version string.
- `#define CH_KERNEL_MAJOR 6`
  
  Kernel version major number.
- `#define CH_KERNEL_MINOR 1`
  
  Kernel version minor number.
- `#define CH_KERNEL_PATCH 4`
  
  Kernel version patch number.

Constants for configuration options

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

Functions

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

7.6.2 Macro Definition Documentation
7.6.2.1 _CHIBIOS_RT_

#define _CHIBIOS_RT_

ChibiOS/RT identification macro.

7.6.2.2 CH_KERNEL_STABLE

#define CH_KERNEL_STABLE 1

Stable release flag.

7.6.2.3 CH_KERNEL_VERSION

#define CH_KERNEL_VERSION "6.1.4"

Kernel version string.

7.6.2.4 CH_KERNEL_MAJOR

#define CH_KERNEL_MAJOR 6

Kernel version major number.

7.6.2.5 CH_KERNEL_MINOR

#define CH_KERNEL_MINOR 1

Kernel version minor number.

7.6.2.6 CH_KERNEL_PATCH

#define CH_KERNEL_PATCH 4

Kernel version patch number.
7.6.2.7 FALSE

#define FALSE 0

Generic 'false' preprocessor boolean constant.

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

7.6.2.8 TRUE

#define TRUE 1

Generic 'true' preprocessor boolean constant.

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

7.6.3 Function Documentation

7.6.3.1 chSysHalt()

void chSysHalt (  
    const char * reason )

Halts the system.

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

Note
Can be invoked from any system state.

Parameters

| in   | reason | pointer to an error string |

Function Class:
Special function, this function has special requirements see the notes.
Here is the call graph for this function:

```
chSysHalt  _trace_halt
```

![Call Graph Diagram](image)
7.7 Configuration

7.7.1 Detailed Description

Modules

- Options
- Checks
- Restrictions
7.8 Options

7.8.1 Detailed Description

Kernel related settings and hooks.

System timers settings

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

Kernel parameters and options

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

Performance options

- \#define CH_CFG_OPTIMIZE_SPEED TRUE
  OS optimization.

Subsystem options

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

OSLIB options

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

Objects factory options

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

Enables factory for mailboxes.
• #define CH_CFG_FACTORY_OBJ_FIFOS TRUE

Enables factory for objects FIFOs.
• #define CH_CFG_FACTORY_PIPES TRUE

Enables factory for Pipes.

Debug options

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

Kernel hooks

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

7.8.2  Macro Definition Documentation

7.8.2.1  CH_CFG_ST_RESOLUTION

\#define CH_CFG_ST_RESOLUTION 32

System time counter resolution.

Note
  Allowed values are 16, 32 or 64 bits.

7.8.2.2  CH_CFG_ST_FREQUENCY

\#define CH_CFG_ST_FREQUENCY 10000

System tick frequency.

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

7.8.2.3  CH_CFG_INTERVALS_SIZE

\#define CH_CFG_INTERVALS_SIZE 32

Time intervals data size.

Note
  Allowed values are 16, 32 or 64 bits.
7.8.2.4 CH_CFG_TIME_TYPES_SIZE

#define CH_CFG_TIME_TYPES_SIZE 32

Time types data size.

Note

Allowed values are 16 or 32 bits.

7.8.2.5 CH_CFG_ST_TIMEDELTA

#define CH_CFG_ST_TIMEDELTA 2

Time delta constant for the tick-less mode.

Note

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

7.8.2.6 CH_CFG_TIME_QUANTUM

#define CH_CFG_TIME_QUANTUM 0

Round robin interval.

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

Note

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

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

7.8.2.7 CH_CFG_NO_IDLE_THREAD

#define CH_CFG_NO_IDLE_THREAD FALSE

Idle thread automatic spawn suppression.

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

7.8.2.8 CH_CFG_OPTIMIZE_SPEED

#define CH_CFG_OPTIMIZE_SPEED TRUE

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

Note
This is not related to the compiler optimization options.
The default is TRUE.

7.8.2.9 CH_CFG_USE_TM

#define CH_CFG_USE_TM TRUE

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

Note
The default is TRUE.

7.8.2.10 CH_CFG_USE_REGISTRY

#define CH_CFG_USE_REGISTRY TRUE

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

Note
The default is TRUE.

7.8.2.11 CH_CFG_USE_WAITEXIT

#define CH_CFG_USE_WAITEXIT TRUE

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

Note
The default is TRUE.
7.8.2.12 **CH_CFG_USE_SEMAPHORES**

```c
#define CH_CFG_USE_SEMAPHORES TRUE
```

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

**Note**

The default is TRUE.

7.8.2.13 **CH_CFG_USE_SEMAPHORES_PRIORITY**

```c
#define CH_CFG_USE_SEMAPHORES_PRIORITY FALSE
```

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

**Note**

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

Requires CH_CFG_USE_SEMAPHORES.

7.8.2.14 **CH_CFG_USE_MUTEXES**

```c
#define CH_CFG_USE_MUTEXES TRUE
```

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

**Note**

The default is TRUE.

7.8.2.15 **CH_CFG_USE_MUTEXES_RECURSIVE**

```c
#define CH_CFG_USE_MUTEXES_RECURSIVE FALSE
```

Enables recursive behavior on mutexes.

**Note**

Recursive mutexes are heavier and have an increased memory footprint.

The default is FALSE.

Requires CH_CFG_USE_MUTEXES.
7.8.2.16  CH_CFG_USE_CONDVARS

#define CH_CFG_USE_CONDVARS TRUE

Conditional Variables APIs.

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

Note

The default is TRUE.
Requires CH_CFG_USE_MUTEXES.

7.8.2.17  CH_CFG_USE_CONDVARS_TIMEOUT

#define CH_CFG_USE_CONDVARS_TIMEOUT TRUE

Conditional Variables APIs with timeout.

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

Note

The default is TRUE.
Requires CH_CFG_USE_CONDVARS.

7.8.2.18  CH_CFG_USE_EVENTS

#define CH_CFG_USE_EVENTS TRUE

Events Flags APIs.

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

Note

The default is TRUE.
7.8.2.19 CH_CFG_USE_EVENTS_TIMEOUT

#define CH_CFG_USE_EVENTS_TIMEOUT TRUE

Events Flags APIs with timeout.

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

Note
The default is TRUE.
Requires CH_CFG_USE_EVENTS.

7.8.2.20 CH_CFG_USE_MESSAGES

#define CH_CFG_USE_MESSAGES TRUE

Synchronous Messages APIs.

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

Note
The default is TRUE.

7.8.2.21 CH_CFG_USE_MESSAGES_PRIORITY

#define CH_CFG_USE_MESSAGES_PRIORITY FALSE

Synchronous Messages queuing mode.

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

Note
The default is FALSE. Enable this if you have special requirements.
Requires CH_CFG_USE_MESSAGES.
7.8 Options

7.8.2.22 CH_CFG_USE_DYNAMIC

#define CH_CFG_USE_DYNAMIC TRUE

Dynamic Threads APIs.

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

Note

The default is TRUE.

Requires CH_CFG_USE_WAITEXIT.

Requires CH_CFG_USE_HEAP and/or CH_CFG_USE_MEMPOOLS.

7.8.2.23 CH_CFG_USE_MAILBOXES

#define CH_CFG_USE_MAILBOXES TRUE

Mailboxes APIs.

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

Note

The default is TRUE.

Requires CH_CFG_USE_SEMAPHORES.

7.8.2.24 CH_CFG_USE_MEMCORE

#define CH_CFG_USE_MEMCORE TRUE

Core Memory Manager APIs.

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

Note

The default is TRUE.
7.8.2.25  CH_CFG_MEMCORE_SIZE

#define CH_CFG_MEMCORE_SIZE 0

Managed RAM size.

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

Note

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

7.8.2.26  CH_CFG_USE_HEAP

#define CH_CFG_USE_HEAP TRUE

Heap Allocator APIs.

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

Note

The default is TRUE.
Requires CH_CFG_USE_MEMCORE and either CH_CFG_USE_MUTEXES or CH_CFG_USE_SEMAPHORE.
Mutexes are recommended.

7.8.2.27  CH_CFG_USE_MEMPOOLS

#define CH_CFG_USE_MEMPOOLS TRUE

Memory Pools Allocator APIs.

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

Note

The default is TRUE.
7.8 Options

7.8.2.28 CH_CFG_USE_OBJ_FIFOS

#define CH_CFG_USE_OBJ_FIFOS TRUE

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

Note
The default is TRUE.

7.8.2.29 CH_CFG_USE_PIPES

#define CH_CFG_USE_PIPES TRUE

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

Note
The default is TRUE.

7.8.2.30 CH_CFG_USE_OBJ_CACHES

#define CH_CFG_USE_OBJ_CACHES TRUE

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

Note
The default is TRUE.

7.8.2.31 CH_CFG_USE_DELEGATES

#define CH_CFG_USE_DELEGATES TRUE

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

Note
The default is TRUE.
7.8.2.32 CH_CFG_USE_JOBS

#define CH_CFG_USE_JOBS TRUE

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

Note
The default is TRUE.

7.8.2.33 CH_CFG_USE_FACTORY

#define CH_CFG_USE_FACTORY TRUE

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

Note
The default is FALSE.

7.8.2.34 CH_CFG_FACTORY_MAX_NAMES_LENGTH

#define CH_CFG_FACTORY_MAX_NAMES_LENGTH 8

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

7.8.2.35 CH_CFG_FACTORY_OBJECTS_REGISTRY

#define CH_CFG_FACTORY_OBJECTS_REGISTRY TRUE

Enables the registry of generic objects.

7.8.2.36 CH_CFG_FACTORY_GENERIC_BUFFERS

#define CH_CFG_FACTORY_GENERIC_BUFFERS TRUE

Enables factory for generic buffers.
7.8 Options

7.8.2.37 CH_CFG_FACTORY_SEMAPHORES

#define CH_CFG_FACTORY_SEMAPHORES TRUE

Enables factory for semaphores.

7.8.2.38 CH_CFG_FACTORY_MAILBOXES

#define CH_CFG_FACTORY_MAILBOXES TRUE

Enables factory for mailboxes.

7.8.2.39 CH_CFG_FACTORY_OBJ_FIFOS

#define CH_CFG_FACTORY_OBJ_FIFOS TRUE

Enables factory for objects FIFOs.

7.8.2.40 CH_CFG_FACTORY_PIPES

#define CH_CFG_FACTORY_PIPES TRUE

Enables factory for Pipes.

7.8.2.41 CH_DBG_STATISTICS

#define CH_DBG_STATISTICS FALSE

Debug option, kernel statistics.

Note

The default is FALSE.
7.8.2.42 CH_DBG_SYSTEM_STATE_CHECK

#define CH_DBG_SYSTEM_STATE_CHECK TRUE

Debug option, system state check.
If enabled the correct call protocol for system APIs is checked at runtime.

Note
The default is FALSE.

7.8.2.43 CH_DBG_ENABLE_CHECKS

#define CH_DBG_ENABLE_CHECKS TRUE

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

Note
The default is FALSE.

7.8.2.44 CH_DBG_ENABLE_ASSERTS

#define CH_DBG_ENABLE_ASSERTS TRUE

Debug option, consistency checks.
If enabled then all the assertions in the kernel code are activated. This includes consistency checks inside the kernel, runtime anomalies and port-defined checks.

Note
The default is FALSE.

7.8.2.45 CH_DBG_TRACE_MASK

#define CH_DBG_TRACE_MASK CH_DBG_TRACE_MASK_ALL

Debug option, trace buffer.
If enabled then the trace buffer is activated.

Note
The default is CH_DBG_TRACE_MASK_DISABLED.
7.8.2.46 CH_DBG_TRACE_BUFFER_SIZE

#define CH_DBG_TRACE_BUFFER_SIZE 128

Trace buffer entries.

Note

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

7.8.2.47 CH_DBG_ENABLE_STACK_CHECK

#define CH_DBG_ENABLE_STACK_CHECK TRUE

Debug option, stack checks.

If enabled then a runtime stack check is performed.

Note

The default is FALSE.

The stack check is performed in a architecture/port dependent way. It may not be implemented or some ports.

The default failure mode is to halt the system with the global panic_msg variable set to NULL.

7.8.2.48 CH_DBG_FILL_THREADS

#define CH_DBG_FILL_THREADS TRUE

Debug option, stacks initialization.

If enabled then the threads working area is filled with a byte value when a thread is created. This can be useful for the runtime measurement of the used stack.

Note

The default is FALSE.
### 7.8.2.49 CH_DBG_THREADS_PROFILING

```c
#define CH_DBG_THREADS_PROFILING FALSE
```

Debug option, threads profiling.

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

**Note**

The default is `FALSE`.

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

### 7.8.2.50 CH_CFG_SYSTEM_EXTRA_FIELDS

```c
#define CH_CFG_SYSTEM_EXTRA_FIELDS /* Add threads custom fields here.*/
```

System structure extension.

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

### 7.8.2.51 CH_CFG_SYSTEM_INIT_HOOK

```c
#define CH_CFG_SYSTEM_INIT_HOOK( )
```

**Value:**

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

System initialization hook.

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

### 7.8.2.52 CH_CFG_THREAD_EXTRA_FIELDS

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

Threads descriptor structure extension.

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

#define CH_CFG_THREAD_INIT_HOOK(tp)

Value:

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

Threads initialization hook.
User initialization code added to the _thread_init() function.

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

7.8.2.54 CH_CFG_THREAD_EXIT_HOOK

#define CH_CFG_THREAD_EXIT_HOOK(tp)

Value:

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

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

7.8.2.55 CH_CFG_CONTEXT_SWITCH_HOOK

#define CH_CFG_CONTEXT_SWITCH_HOOK(ntp, otp)

Value:

{ /* Context switch code here. */
}

Context switch hook.
This hook is invoked just before switching between threads.

7.8.2.56 CH_CFG_IRQ_PROLOGUE_HOOK

#define CH_CFG_IRQ_PROLOGUE_HOOK()

Value:

{ /* IRQ prologue code here. */
}

ISR enter hook.
7.8.2.57 CH_CFG_IRQ_EPILOGUE_HOOK

#define CH_CFG_IRQ_EPILOGUE_HOOK( )

Value:
{
    /* IRQ epilogue code here.*/
}

ISR exit hook.

7.8.2.58 CH_CFG_IDLE_ENTER_HOOK

#define CH_CFG_IDLE_ENTER_HOOK( )

Value:
{
    /* Idle-enter code here.*/
}

Idle thread enter hook.

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

7.8.2.59 CH_CFG_IDLE_LEAVE_HOOK

#define CH_CFG_IDLE_LEAVE_HOOK( )

Value:
{
    /* Idle-leave code here.*/
}

Idle thread leave hook.

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

7.8.2.60 CH_CFG_IDLE_LOOP_HOOK

#define CH_CFG_IDLE_LOOP_HOOK( )

Value:
/* Idle loop code here. */

Idle Loop hook.
This hook is continuously invoked by the idle thread loop.

7.8.2.61 CH_CFG_SYSTEM_TICK_HOOK

#define CH_CFG_SYSTEM_TICK_HOOK( )

Value:
/* System tick event code here. */

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

7.8.2.62 CH_CFG_SYSTEM_HALT_HOOK

#define CH_CFG_SYSTEM_HALT_HOOK( reason )

Value:
/* System halt code here. */

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

7.8.2.63 CH_CFG_TRACE_HOOK

#define CH_CFG_TRACE_HOOK( tep )

Value:
/* Trace code here. */

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

This module performs a series of checks on configuration data, it is able to detect and reject obsolete or incomplete `chconf.h` files.
7.10 Restrictions

7.10.1 Detailed Description

This module is responsible for applying license-related restrictions to the configuration options.
7.11 Base Kernel Services

7.11.1 Detailed Description

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

Modules

- Memory Alignment
- System Management
- Lists and Queues
- Scheduler
- Time and Intervals
- Virtual Timers
- Threads
- Time Measurement
7.12 Memory Alignment

7.12.1 Detailed Description

Memory Alignment services.

Memory alignment support macros

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

7.12.2 Macro Definition Documentation

7.12.2.1 MEM_ALIGN_MASK

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

Alignment mask constant.

Parameters

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

7.12.2.2 MEM_ALIGN_PREV

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

Value:

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

Aligns to the previous aligned memory address.
Parameters

<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>

7.12.2.3 MEM_ALIGN_NEXT

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

Value:

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

Aligns to the next aligned memory address.

Parameters

<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>

7.12.2.4 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>

7.12.2.5 MEM_IS_VALID_ALIGNMENT

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

Returns whatever a constant is a valid alignment.

Valid alignments are powers of two.
Parameters

| in | a | alignment to be checked, must be a constant |
7.13 System Management

7.13.1 Detailed Description

System related APIs and services:

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

Macros

- #define chSysGetRealtimeCounterX() (rtcnt_t)port_rt_get_counter_value()
  
  Returns the current value of the system real time counter.
- #define chSysSwitch(ntp, otp)
  
  Performs a context switch.

Masks of executable integrity checks.

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

ISRs abstraction macros

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

Fast ISRs abstraction macros

- #define CH_FAST_IRQ_HANDLER(id) PORT_FAST_IRQ_HANDLER(id)
  
  Standard fast IRQ handler declaration.
Time conversion utilities for the realtime counter

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

Functions

- `THD_WORKING_AREA (ch_idle_thread_wa, PORT_IDLE_THREAD_STACK_SIZE)`
  Idle thread working area.
- `static void _idle_thread (void *)`
  This function implements the idle thread infinite loop.
- `void chSysInit (void)`
  ChibiOS/RT initialization.
- `void chSysHalt (const char *)`
  Halts the system.
- `bool chSysIntegrityCheckI (unsigned testmask)`
  System integrity check.
- `void chSysTimerHandlerI (void)`
  Handles time ticks for round robin preemption and timer increments.
- `syssts_t chSysGetStatusAndLockX (void)`
  Returns the execution status and enters a critical zone.
- `void chSysRestoreStatusX (syssts_t sts)`
  Restores the specified execution status and leaves a critical zone.
- `bool chSysIsCounterWithinX (rtcnt_t cnt, rtcnt_t start, rtcnt_t end)`
  Realtime window test.
- `void chSysPolledDelayX (rtcnt_t cycles)`
  Polled delay.
- `static void chSysDisable (void)`
  Raises the system interrupt priority mask to the maximum level.
- `static void chSysSuspend (void)`
  Raises the system interrupt priority mask to system level.
- `static void chSysEnable (void)`
  Lowers the system interrupt priority mask to user level.
- `static void chSysLock (void)`
  Enters the kernel lock state.
- `static void chSysUnlock (void)`
  Leaves the kernel lock state.
- `static void chSysLockFromISR (void)`
  Enters the kernel lock state from within an interrupt handler.
- `static void chSysUnlockFromISR (void)`
  Leaves the kernel lock state from within an interrupt handler.
Leaves the kernel lock state from within an interrupt handler.

- static void chSysUnconditionalLock (void)
  Unconditionally enters the kernel lock state.
- static void chSysUnconditionalUnlock (void)
  Unconditionally leaves the kernel lock state.
- static thread_t * chSysGetIdleThreadX (void)
  Returns a pointer to the idle thread.

7.13.2 Macro Definition Documentation

7.13.2.1 CH_IRQ_IS_VALID_PRIORITY

#define CH_IRQ_IS_VALID_PRIORITY(
  prio ) PORT_IRQ_IS_VALID_PRIORITY(prio)

Priority level validation macro.

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

Parameters

| in  | 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. |

7.13.2.2 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 |

ChibiOS/RT
Returns

Priority range result.

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>false</td>
<td>if the priority is invalid or if the architecture does not support priorities.</td>
</tr>
<tr>
<td>true</td>
<td>if the priority is valid.</td>
</tr>
</tbody>
</table>

7.13.2.3 CH_IRQ_PROLOGUE

#define CH_IRQ_PROLOGUE()

Value:

PORT_IRQ_PROLOGUE();
CH_CFG_IRQ_PROLOGUE_HOOK();
_stats_increase_irq();
_trace_isr_enter(__func__);
_dbg_check_enter_isr()

IRQ handler enter code.

Note

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

Function Class:

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

7.13.2.4 CH_IRQ_EPILOGUE

#define CH_IRQ_EPILOGUE()

Value:

_dbg_check_leave_isr();
_trace_isr_leave(__func__);
CH_CFG_IRQ_EPILOGUE_HOOK();
PORT_IRQ_EPILOGUE()

IRQ handler exit code.

Note

Usually IRQ handlers function are also declared naked.
This macro usually performs the final reschedule by using chSchIsPreemptionRequired() and chSchDoReschedule().

Function Class:

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

#define CH_IRQ_HANDLER(id) PORT_IRQ_HANDLER(id)

Standard normal IRQ handler declaration.

**Note**

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

**Function Class:**

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

7.13.2.6 CH_FAST_IRQ_HANDLER

#define CH_FAST_IRQ_HANDLER(id) PORT_FAST_IRQ_HANDLER(id)

Standard fast IRQ handler declaration.

**Note**

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

Not all architectures support fast interrupts.

**Function Class:**

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

7.13.2.7 S2RTC

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

Seconds to realtime counter.

Converts from seconds to realtime counter cycles.

**Note**

The macro assumes that *freq* \(\geq 1\).
Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>freq</th>
<th>clock frequency, in Hz, of the realtime counter</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>sec</td>
<td>number of seconds</td>
</tr>
</tbody>
</table>

Returns

The number of cycles.

Function Class:

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

7.13.2.8 MS2RTC

#define MS2RTC(f, m) (rtcnt_t)((((f) + 999UL) / 1000UL) * (m))

Milliseconds to realtime counter.

Converts from milliseconds to realtime counter cycles.

Note

The result is rounded upward to the next millisecond boundary.

The macro assumes that \( f \geq 1000 \).

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>freq</th>
<th>clock frequency, in Hz, of the realtime counter</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>msec</td>
<td>number of milliseconds</td>
</tr>
</tbody>
</table>

Returns

The number of cycles.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.
### 7.13.2.9 US2RTC

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

Microseconds to realtime counter.

Converts from microseconds to realtime counter cycles.

**Note**

The result is rounded upward to the next microsecond boundary.

The macro assumes that `freq >= 1000000`.

**Parameters**

| in   | freq, usec | clock frequency, in Hz, of the realtime counter | number of microseconds |

**Returns**

The number of cycles.

**Function Class:**

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

### 7.13.2.10 RTC2S

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

Realtime counter cycles to seconds.

Converts from realtime counter cycles number to seconds.

**Note**

The result is rounded up to the next second boundary.

The macro assumes that `freq >= 1`.

**Parameters**

| in   | freq, n | clock frequency, in Hz, of the realtime counter | number of cycles |

7.13 System Management

Returns

The number of seconds.

Function Class:

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

7.13.2.11 RTC2MS

```c
#define RTC2MS(
    freq,
    n ) (((n) - 1UL) / ((freq) / 1000UL)) + 1UL
```

Converts from realtime counter cycles number to milliseconds.

Note

The result is rounded up to the next millisecond boundary.

The macro assumes that freq >= 1000.

Parameters

<table>
<thead>
<tr>
<th>in freq</th>
<th>clock frequency, in Hz, of the realtime counter</th>
</tr>
</thead>
<tbody>
<tr>
<td>in n</td>
<td>number of cycles</td>
</tr>
</tbody>
</table>

Returns

The number of milliseconds.

Function Class:

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

7.13.2.12 RTC2US

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

Converts from realtime counter cycles number to microseconds.

Note

The result is rounded up to the next microsecond boundary.

The macro assumes that freq >= 1000000.
### Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>freq</th>
<th>clock frequency, in Hz, of the realtime counter</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>n</td>
<td>number of cycles</td>
</tr>
</tbody>
</table>

### Returns

The number of microseconds.

### Function Class:

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

### 7.13.2.13 chSysGetRealtimeCounterX

```c
#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.

### 7.13.2.14 chSysSwitch

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

**Value:**

```c
\ 
_trace_switch(ntp, otp); 
_stats_ctxswc(ntp, otp); 
CH_CFG_CONTEXT_SWITCH_HOOK(ntp, otp); 
port_switch(ntp, otp); 
```

Performs a context switch.

**Note**

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

Parameters

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>ntp</td>
<td>the thread to be switched in</td>
</tr>
<tr>
<td>in</td>
<td>otp</td>
<td>the thread to be switched out</td>
</tr>
</tbody>
</table>

Function Class:

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

7.13.3 Function Documentation

7.13.3.1 THD_WORKING_AREA()

THD_WORKING_AREA (
    ch_idle_thread_wa ,
    PORT_IDLE_THREAD_STACK_SIZE )

Idle thread working area.

7.13.3.2 _idle_thread()

static void _idle_thread ( 
    void * p ) [static]

This function implements the idle thread infinite loop.

The function puts the processor in the lowest power mode capable to serve interrupts.
The priority is internally set to the minimum system value so that this thread is executed only if there are no other ready threads in the system.

Parameters

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>p</td>
<td>the thread parameter, unused in this scenario</td>
</tr>
</tbody>
</table>

7.13.3.3 chSysInit()

void chSysInit ( 
    void 
)

ChibiOS/RT initialization.

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

Interrupts must disabled before invoking this function.

Postcondition

The main thread is created with priority \texttt{NORMALPRIO} and interrupts are enabled.

Function Class:

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

Here is the call graph for this function:
7.13.3.4 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.

Here is the call graph for this function:

```
chSysHalt → _trace_halt
```

7.13.3.5 chSysIntegrityCheckI()

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

System integrity check.

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

Note

The appropriate action in case of failure is to halt the system before releasing the critical zone. If the system is corrupted then one possible outcome of this function is an exception caused by NULL or corrupted pointers in list elements. Exception vectors must be monitored as well. This function is not used internally, it is up to the application to define if and where to perform system checking. Performing all tests at once can be a slow operation and can degrade the system response time. It is suggested to execute one test at time and release the critical zone in between tests.
Parameters

| in   | testmask | Each bit in this mask is associated to a test to be performed. |

Returns

The test result.

Return values

| false | The test succeeded. |
| true  | Test failed. |

Function Class:

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

Here is the call graph for this function:

```
chSysIntegrityCheckI chDbgCheckClassI chSysHalt
```

7.13.3.6 chSysTimerHandlerI()

```c
void chSysTimerHandlerI()
{
    void
}
```

Handles time ticks for round robin preemption and timer increments.

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

Note

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

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

Here is the call graph for this function:

```
chSysTimerHandlerI
    chDbgCheckClassI
    chVTDoTickI
    chSysHalt
    chSysLockFromISR
    chSysUnlockFromISR
    chTimeAddX
    chTimeDiffX
    chVTGetSystemTimeX
    vt_dequeue
    vt_is_empty
    vt_set_alarm
```

### 7.13.3.7 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.

Here is the call graph for this function:

```
chSysGetStatusAndLockX
chSysLock
chSysLockFromISR
_dbg_check_lock
_stats_start_measure
_crit_thd
_dbg_check_lock_from_isr
_stats_start_measure
_crit_isr
```

### 7.13.3.8 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. |
## 7.13 System Management

### Function Class:

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

Here is the call graph for this function:

### 7.13.3.9 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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>cnt</code></td>
<td>the counter value to be tested</td>
</tr>
<tr>
<td><code>start</code></td>
<td>the start of the time window (inclusive)</td>
</tr>
<tr>
<td><code>end</code></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.

### 7.13.3.10 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:

![Call graph](image)

### 7.13.3.11 chSysDisable()

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

Raises the system interrupt priority mask to the maximum level.

All the maskable interrupt sources are disabled regardless their hardware priority.
Note
Do not invoke this API from within a kernel lock.

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

Here is the call graph for this function:

```
  chSysDisable  _dbg_check_disable  chSysHalt
```

7.13.3.12 chSysSuspend()

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

Raises the system interrupt priority mask to system level.

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

Note
Do not invoke this API from within a kernel lock.

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

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

Here is the call graph for this function:

```
  chSysSuspend  _dbg_check_suspend  chSysHalt
```
### 7.13.3.13 chSysEnable()

```c
static void chSysEnable {
    void ) [inline], [static]
```

Lowers the system interrupt priority mask to user level.

All the interrupt sources are enabled.

**Note**

Do not invoke this API from within a kernel lock.

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

**Function Class:**

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

Here is the call graph for this function:

![Call Graph for chSysEnable]

### 7.13.3.14 chSysLock()

```c
static void chSysLock {
    void ) [inline], [static]
```

Enters the kernel lock state.

**Function Class:**

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

Here is the call graph for this function:

![Call Graph for chSysLock]
7.13.3.15 **chSysUnlock()**

```c
static void chSysUnlock (
    void ) [inline], [static]
```
Leaves the kernel lock state.

Function Class:

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

Here is the call graph for this function:

![Call Graph for chSysUnlock]  

7.13.3.16 **chSysLockFromISR()**

```c
static void chSysLockFromISR (
    void ) [inline], [static]
```
Enters the kernel lock state from within an interrupt handler.

Note

This API may do nothing on some architectures, it is required because on ports that support preemptable interrupt handlers it is required to raise the interrupt mask to the same level of the system mutual exclusion zone.  
It is good practice to invoke this API before invoking any I-class syscall from an interrupt handler.  
This API must be invoked exclusively from interrupt handlers.

Function Class:

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

Here is the call graph for this function:

![Call Graph for chSysLockFromISR]
7.13.3.17  chSysUnlockFromISR()

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

Leaves the kernel lock state from within an interrupt handler.

**Note**

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

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

This API must be invoked exclusively from interrupt handlers.

**Function Class:**

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

Here is the call graph for this function:

![Call graph for chSysUnlockFromISR](ch_graph.png)

7.13.3.18  chSysUnconditionalLock()

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

Unconditionally enters the kernel lock state.

**Note**

Can be called without previous knowledge of the current lock state. The final state is "s-locked".
Function Class:
Special function, this function has special requirements see the notes.

Here is the call graph for this function:

```
chSysUnconditionalLock  
<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>chSysUnlock</td>
<td></td>
</tr>
<tr>
<td></td>
<td>_dbg_check_lock</td>
</tr>
<tr>
<td></td>
<td>_stats_start_measure</td>
</tr>
<tr>
<td></td>
<td>_crit_thd</td>
</tr>
</tbody>
</table>
```

### 7.13.3.19 chSysUnconditionalUnlock()

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

Unconditionally leaves the kernel lock state.

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

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

Here is the call graph for this function:

```
chSysUnconditionalUnlock  
<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>chSysUnlock</td>
<td></td>
</tr>
<tr>
<td></td>
<td>_dbg_check_unlock</td>
</tr>
<tr>
<td></td>
<td>_stats_stop_measure</td>
</tr>
<tr>
<td></td>
<td>_crit_thd</td>
</tr>
</tbody>
</table>
```
7.13.3.20 chSysGetIdleThreadX()

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

Returns a pointer to the idle thread.

Precondition

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

Note

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

Returns

   Pointer to the idle thread.

Function Class:

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

7.14.1 Detailed Description

Macros

- `#define _CH_QUEUE_DATA(name) ((ch_queue_t *)&name, (ch_queue_t *)&name)`
  Data part of a static queue object initializer.
- `#define CH_QUEUE_DECL(name) ch_queue_t name = _CH_QUEUE_DATA(name)`
  Static queue object initializer.

Typedefs

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

Data Structures

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

Functions

- `static void ch_list_init (ch_list_t *lp)`
  List initialization.
- `static bool ch_list_isempty (ch_list_t *lp)`
  Evaluates to true if the specified list is empty.
- `static bool ch_list_notempty (ch_list_t *lp)`
  Evaluates to true if the specified list is not empty.
- `static void ch_list_push (ch_list_t *p, ch_list_t *lp)`
  Pushes an element on top of a stack list.
- `static ch_list_t * ch_list_pop (ch_list_t *lp)`
  Pops an element from the top of a stack list and returns it.
- `static void ch_queue_init (ch_queue_t *qp)`
  Queue initialization.
- `static bool ch_queue_isempty (const ch_queue_t *qp)`
  Evaluates to true if the specified queue is empty.
- `static bool ch_queue_notempty (const ch_queue_t *qp)`
  Evaluates to true if the specified queue is not empty.
- `static void ch_queue_insert (ch_queue_t *p, ch_queue_t *qp)`
Inserts an element into a queue.
- static ch_queue_t * ch_queue_fifo_remove (ch_queue_t *qp)
  Removes the first-out element from a queue and returns it.
- static ch_queue_t * ch_queue_lifo_remove (ch_queue_t *qp)
  Removes the last-out element from a queue and returns it.
- static ch_queue_t * ch_queue_dequeue (ch_queue_t *p)
  Removes an element from a queue and returns it.
- static void ch_pqueue_init (ch_priority_queue_t *pqp)
  Priority queue initialization.
- static ch_priority_queue_t * ch_pqueue_remove_highest (ch_priority_queue_t *pqp)
  Removes the highest priority element from a priority queue and returns it.
- static ch_priority_queue_t * ch_pqueue_insert_behind (ch_priority_queue_t *pqp, ch_priority_queue_t *p)
  Inserts an element in the priority queue placing it behind its peers.
- static ch_priority_queue_t * ch_pqueue_insert_ahead (ch_priority_queue_t *pqp, ch_priority_queue_t *p)
  Inserts an element in the priority queue placing it ahead of its peers.

7.14.2 Macro Definition Documentation

7.14.2.1 _CH_QUEUE_DATA

#define _CH_QUEUE_DATA(
    name ) { (ch_queue_t *) &name, (ch_queue_t *) &name }

Data part of a static queue object initializer.

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

Parameters

- in name the name of the queue variable

7.14.2.2 CH_QUEUE_DECL

#define CH_QUEUE_DECL(
    name ) ch_queue_t name = _CH_QUEUE_DATA(name)

Static queue object initializer.

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

Parameters

- in name the name of the queue variable
7.14 Lists and Queues

7.14.3 Typedef Documentation

7.14.3.1 ch_list_t

typedef struct ch_list ch_list_t

Type of a generic single link list header and element.

7.14.3.2 ch_queue_t

typedef struct ch_queue ch_queue_t

Type of a generic bidirectional linked list header and element.

7.14.3.3 ch_priority_queue_t

typedef struct ch_priority_queue ch_priority_queue_t

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

7.14.4 Function Documentation

7.14.4.1 ch_list_init()

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

List initialization.

Parameters

| out | lp | pointer to the list header |

Function Class:

Not an API, this function is for internal use only.
7.14.4.2  ch_list_isempty()

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

Evaluates to true if the specified list is empty.

Parameters

| in | lp | pointer to the list header |

Returns

The status of the list.

Function Class:

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

7.14.4.3  ch_list_notempty()

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

Evaluates to true if the specified list is not empty.

Parameters

| in | lp | pointer to the list header |

Returns

The status of the list.

Function Class:

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

7.14.4.4  ch_list_push()

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

Pushes an element on top of a stack list.
### 7.14 Lists and Queues

#### Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>p</th>
<th>the pointer to the element to be inserted in the list</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>lp</td>
<td>the pointer to the list header</td>
</tr>
</tbody>
</table>

**Function Class:**

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

#### 7.14.4.5 ch_list_pop()

```c
static ch_list_t* ch_list_pop ( ch_list_t * lp ) [inline], [static]
```

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

**Precondition**

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

**Parameters**

| in  | lp | the pointer to the list header |

**Returns**

The removed element pointer.

**Function Class:**

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

#### 7.14.4.6 ch_queue_init()

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

Queue initialization.

**Parameters**

| out  | qp | pointer to the queue header |

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

7.14.4.7  ch_queue_isempty()

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

Evaluates to true if the specified queue is empty.

Parameters
   **qp**  pointer to the queue header

Returns
   The status of the queue.

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

7.14.4.8  ch_queue_notempty()

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

Evaluates to true if the specified queue is not empty.

Parameters
   **qp**  pointer to the queue header

Returns
   The status of the queue.

Function Class:
   Not an API, this function is for internal use only.
7.14.4.9 ch_queue_insert()

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

Inserts an element into a queue.

**Parameters**

- **p** the pointer to the element to be inserted in the queue
- **qp** the pointer to the queue header

**Function Class:**

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

7.14.4.10 ch_queue_fifo_remove()

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

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

**Note**

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

**Parameters**

- **qp** the pointer to the queue list header

**Returns**

The removed element pointer.

**Function Class:**

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

7.14.4.11 ch_queue_lifo_remove()

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

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

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

Parameters

| in | qp | the pointer to the queue list header |

Returns

The removed element pointer.

Function Class:

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

7.14.4.12 ch_queue_dequeue()

static ch_queue_t* ch_queue_dequeue {
    ch_queue_t * p } [inline], [static]

Removes an element from a queue and returns it.

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

Parameters

| in | p | the pointer to the element to be removed from the queue |

Returns

The removed element pointer.

Function Class:

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

7.14.4.13 ch_pqueue_init()

static void ch_pqueue_init {
    ch_priority_queue_t * pqp } [inline], [static]

Priority queue initialization.

Note

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

| out | pqp | pointer to the priority queue header |

Function Class:

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

### 7.14.4.14 ch_pqueue_remove_highest()

```c
static ch_priority_queue_t* ch_pqueue_remove_highest ( ch_priority_queue_t * pqp );
```

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

Parameters

| in | pqp | the pointer to the priority queue list header |

Returns

The removed element pointer.

Function Class:

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

### 7.14.4.15 ch_pqueue_insert_behind()

```c
static ch_priority_queue_t* ch_pqueue_insert_behind ( ch_priority_queue_t * pqp, ch_priority_queue_t * p );
```

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

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

Parameters

| in | pqp | the pointer to the priority queue list header |
| in | p | the pointer to the element to be inserted in the queue |
Returns

The inserted element pointer.

Function Class:

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

7.14.4.16 ch_pqueue_insert_ahead()

\[
\text{static ch_priority_queue_t* ch_pqueue_insert_ahead (}
\quad \text{ch_priority_queue_t* pqp,}
\quad \text{ch_priority_queue_t* p) [inline], [static]}
\]

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

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

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>pqp</th>
<th>the pointer to the priority queue list header</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>p</td>
<td>the pointer to the element to be inserted in the queue</td>
</tr>
</tbody>
</table>

Returns

The inserted element pointer.

Function Class:

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

7.15.1 Detailed Description

This module provides the default portable scheduler code.

Macros

- `#define firstprio(rlp) ((rlp)->next->prio)`
  Returns the priority of the first thread on the given ready list.
- `#define currp ch.rlist.current`
  Current thread pointer access macro.
- `#define __CH_STRINGIFY(a) #a`
  Utility to make the parameter a quoted string.

Wakeup status codes

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

Priority constants

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

- `#define CH_STATE_READY (tstate_t)0`
  Waiting on the ready list.

- `#define CH_STATE_CURRENT (tstate_t)1`
  Currently running.

- `#define CH_STATE_WTSTART (tstate_t)2`
  Just created.

- `#define CH_STATE_SUSPENDED (tstate_t)3`
  Suspended state.

- `#define CH_STATE_QUEUE (tstate_t)4`
  On a queue.

- `#define CH_STATE_WTSEM (tstate_t)5`
  On a semaphore.

- `#define CH_STATE_WTMTX (tstate_t)6`
  On a mutex.

- `#define CH_STATE_WTCOND (tstate_t)7`
  On a cond.variable.

- `#define CH_STATE_SLEEPING (tstate_t)8`
  Sleeping.

- `#define CH_STATE_WTEXIT (tstate_t)9`
  Waiting a thread.

- `#define CH_STATE_WTREVT (tstate_t)10`
  One event.

- `#define CH_STATE_WTANDEVT (tstate_t)11`
  Several events.

- `#define CH_STATE_SNDMSGQ (tstate_t)12`
  Sending a message, in queue.

- `#define CH_STATE_SNDMSG (tstate_t)13`
  Sent a message, waiting answer.

- `#define CH_STATE_WTMSG (tstate_t)14`
  Waiting for a message.

- `#define CH_STATE_FINAL (tstate_t)15`
  Thread terminated.

- `#define CH_STATE_NAMES`
  Thread states as array of strings.
Thread flags and attributes

- `#define CH_FLAG_MODE_MASK (tmode_t)3U`
  Thread memory mode mask.

- `#define CH_FLAG_MODE_STATIC (tmode_t)0U`
  Static thread.

- `#define CH_FLAG_MODE_HEAP (tmode_t)1U`
  Thread allocated from a Memory Heap.

- `#define CH_FLAG_MODE_MPOOL (tmode_t)2U`
  Thread allocated from a Memory Pool.

- `#define CH_FLAG_TERMINATE (tmode_t)4U`
  Termination requested flag.

Typedefs

- `typedef struct ch_delta_list delta_list_t`
  Type of a Virtual Timer structure.

- `typedef struct ch_thread thread_t`
  Type of a thread structure.

- `typedef thread_t * thread_reference_t`
  Type of a thread reference.

- `typedef struct ch_threads_list threads_list_t`
  Type of a generic threads single link list, it works like a stack.

- `typedef struct ch_threads_queue threads_queue_t`
  Type of a generic threads bidirectional linked list header and element.

- `typedef struct ch_ready_list ready_list_t`
  Type of a ready list header.

- `typedef void(* vfunc_t) (void *p)`
  Type of a Virtual Timer callback function.

- `typedef struct ch_virtual_timer virtual_timer_t`
  Type of a Virtual Timer structure.

- `typedef struct ch_virtual_timers_list virtual_timers_list_t`
  Type of virtual timers list header.

- `typedef struct ch_system_debug system_debug_t`
  Type of a system debug structure.

- `typedef struct ch_system ch_system_t`
  Type of system data structure.

Data Structures

- `struct ch_threads_queue`
  Structure representing a threads queue.

- `struct ch_thread`
  Structure representing a thread.

- `struct ch_delta_list`
Virtual Timer delta list element and header structure.

- struct ch_virtual_timer
  Structure representing a Virtual Timer.
- struct ch_virtual_timers_list
  Structure representing a virtual timers list header.
- struct ch_system_debug
  System debug data structure.
- struct ch_system
  System data structure.

Functions

- void ch_sch_prio_insert (ch_queue_t *tp, ch_queue_t *qp)
  Inserts a thread into a priority ordered queue.
- void _scheduler_init (void)
  Scheduler initialization.
- thread_t *chSchReadyI (thread_t *tp)
  Inserts a thread in the Ready List placing it behind its peers.
- thread_t *chSchReadyAheadI (thread_t *tp)
  Inserts a thread in the Ready List placing it ahead its peers.
- void chSchGoSleepS (tstate_t newstate)
  Puts the current thread to sleep into the specified state.
- msg_t chSchGoSleepTimeoutS (tstate_t newstate, sysinterval_t timeout)
  Puts the current thread to sleep into the specified state with timeout specification.
- void chSchWakeupS (thread_t *ntp, msg_t msg)
  Wakes up a thread.
- void chSchRescheduleS (void)
  Performs a reschedule if a higher priority thread is runnable.
- bool chSchIsPreemptionRequired (void)
  Evaluates if preemption is required.
- void chSchDoRescheduleBehind (void)
  Switches to the first thread on the runnable queue.
- void chSchDoRescheduleAhead (void)
  Switches to the first thread on the runnable queue.
- void chSchDoReschedule (void)
  Switches to the first thread on the runnable queue.
- static bool chSchIsRescRequiredI (void)
  Determines if the current thread must reschedule.
- static bool chSchCanYieldS (void)
  Determines if yielding is possible.
- static void chSchDoYieldS (void)
  Yields the time slot.
- static void chSchPreemption (void)
  Inline-able preemption code.

Variables

- ch_system_t ch
  System data structures.
7.15 Scheduler

7.15.2 Macro Definition Documentation

7.15.2.1 MSG_OK

#define MSG_OK (msg_t)0

Normal wakeup message.

7.15.2.2 MSG_TIMEOUT

#define MSG_TIMEOUT (msg_t)-1

Wakeup caused by a timeout condition.

7.15.2.3 MSG_RESET

#define MSG_RESET (msg_t)-2

Wakeup caused by a reset condition.

7.15.2.4 NOPRIOR

#define NOPRIOR (tprio_t)0

Ready list header priority.

7.15.2.5 IDLEPRIOR

#define IDLEPRIOR (tprio_t)1

Idle priority.
7.15.2.6 LOWPRIO

#define LOWPRIO (tprio_t)2

Lowest priority.

7.15.2.7 NORMALPRIO

#define NORMALPRIO (tprio_t)128

Normal priority.

7.15.2.8 HIGHPRIO

#define HIGHPRIO (tprio_t)255

Highest priority.

7.15.2.9 CH_STATE_READY

#define CH_STATE_READY (tstate_t)0

Waiting on the ready list.

7.15.2.10 CH_STATE_CURRENT

#define CH_STATE_CURRENT (tstate_t)1

Currently running.

7.15.2.11 CH_STATE_WTSTART

#define CH_STATE_WTSTART (tstate_t)2

Just created.
7.15 Scheduler

7.15.2.12 CH_STATE_SUSPENDED

#define CH_STATE_SUSPENDED (tstate_t)3

Suspended state.

7.15.2.13 CH_STATE_QUEUED

#define CH_STATE_QUEUED (tstate_t)4

On a queue.

7.15.2.14 CH_STATE_WTSEM

#define CH_STATE_WTSEM (tstate_t)5

On a semaphore.

7.15.2.15 CH_STATE_WTMTX

#define CH_STATE_WTMTX (tstate_t)6

On a mutex.

7.15.2.16 CH_STATE_WTCOND

#define CH_STATE_WTCOND (tstate_t)7

On a cond variable.

7.15.2.17 CH_STATE_SLEEPING

#define CH_STATE_SLEEPING (tstate_t)8

Sleeping.
7.15.2.18 CH_STATE_WTEXIT

#define CH_STATE_WTEXIT (tstate_t)9
Waiting a thread.

7.15.2.19 CH_STATE_WTOREVT

#define CH_STATE_WTOREVT (tstate_t)10
One event.

7.15.2.20 CH_STATE_WTANDEVT

#define CH_STATE_WTANDEVT (tstate_t)11
Several events.

7.15.2.21 CH_STATE_SNDMSGQ

#define CH_STATE_SNDMSGQ (tstate_t)12
Sending a message, in queue.

7.15.2.22 CH_STATE_SNDMSG

#define CH_STATE_SNDMSG (tstate_t)13
Sent a message, waiting answer.

7.15.2.23 CH_STATE_WTMSG

#define CH_STATE_WTMSG (tstate_t)14
Waiting for a message.
7.15.2.24  CH_STATE_FINAL

#define CH_STATE_FINAL (tstate_t)15

Thread terminated.

7.15.2.25  CH_STATE_NAMES

#define CH_STATE_NAMES

Value:
"READY", "CURRENT", "WTSTART", "SUSPENDED", "QUEUED", "WTSEM", "WTMTX", \ 
"WTCOND", "SLEEPING", "WTEXIT", "WTREVT", "WTANDEVT", "SNDMSGQ", \ 
"SNDMSG", "WMSG", "FINAL"

Thread states as array of strings.
Each element in an array initialized with this macro can be indexed using the numeric thread state values.

7.15.2.26  CH_FLAG_MODE_MASK

#define CH_FLAG_MODE_MASK (tmode_t)3U

Thread memory mode mask.

7.15.2.27  CH_FLAG_MODE_STATIC

#define CH_FLAG_MODE_STATIC (tmode_t)0U

Static thread.

7.15.2.28  CH_FLAG_MODE_HEAP

#define CH_FLAG_MODE_HEAP (tmode_t)1U

Thread allocated from a Memory Heap.
7.15.2.29 CH_FLAG_MODE_MPOOL

#define CH_FLAG_MODE_MPOOL (tmode_t)2U

Thread allocated from a Memory Pool.

7.15.2.30 CH_FLAG_TERMINATE

#define CH_FLAG_TERMINATE (tmode_t)4U

Termination requested flag.

7.15.2.31 firstprio

#define firstprio( rlp ) ((rlp)->next->prio)

Returns the priority of the first thread on the given ready list.

Function Class:

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

7.15.2.32 currp

#define currp ch.rlist.current

Current thread pointer access macro.

Note

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

7.15.2.33 __CH_STRINGIFY

#define __CH_STRINGIFY( a ) #a

Utility to make the parameter a quoted string.
7.15 Scheduler

7.15.3 Typedef Documentation

7.15.3.1 delta_list_t

typedef struct ch_delta_list delta_list_t

Type of a Virtual Timer structure.

7.15.3.2 thread_t

typedef struct ch_thread thread_t

Type of a thread structure.

7.15.3.3 thread_reference_t

typedef thread_t* thread_reference_t

Type of a thread reference.

7.15.3.4 threads_list_t

typedef struct ch_threads_list threads_list_t

Type of a generic threads single link list, it works like a stack.

7.15.3.5 threads_queue_t

typedef struct ch_threads_queue threads_queue_t

Type of a generic threads bidirectional linked list header and element.
7.15.3.6 ready_list_t

typedef struct ch_ready_list ready_list_t
Type of a ready list header.

7.15.3.7 vtfunc_t

typedef void(* vtfunc_t) (void *p)
Type of a Virtual Timer callback function.

7.15.3.8 virtual_timer_t

typedef struct ch_virtual_timer virtual_timer_t
Type of a Virtual Timer structure.

7.15.3.9 virtual_timers_list_t

typedef struct ch_virtual_timers_list virtual_timers_list_t
Type of virtual timers list header.

7.15.3.10 system_debug_t

typedef struct ch_system_debug system_debug_t
Type of a system debug structure.

7.15.3.11 ch_system_t

typedef struct ch_system ch_system_t
Type of system data structure.

7.15.4 Function Documentation

7.15.4.1 ch_sch_prio_insert()

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

Inserts a thread into a priority ordered queue.

Note

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>in</strong> tp</td>
<td>the pointer to the thread to be inserted in the list</td>
</tr>
<tr>
<td><strong>in</strong> qp</td>
<td>the pointer to the threads list header</td>
</tr>
</tbody>
</table>

Function Class:

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

### 7.15.4.2 _scheduler_init()

```c
void _scheduler_init ( void )
```

Scheduler initialization.

Function Class:

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

Here is the call graph for this function:

![Call Graph](image)

### 7.15.4.3 chSchReadyI()

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

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

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

**Precondition**

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

**Postcondition**

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

\[
\text{in } \text{tp} \text{ the thread to be made ready}
\]

Returns

The thread pointer.

Function Class:

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

Here is the call graph for this function:

```
chSchReadyI
    ch_pqueue_insert_behind
    chSchReadyI
    chDbgCheckClassI
    chSysHalt
```

7.15.4.4 chSchReadyAheadI()

\[
\text{thread_t } \ast \text{chSchReadyAheadI (}
\text{thread_t } \ast \text{tp }
\)
\]

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

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

Precondition

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

Postcondition

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

| in  | tp | the thread to be made ready |

Returns

The thread pointer.

Function Class:

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

Here is the call graph for this function:

7.15.4.5 chSchGoSleepS()

```c
void chSchGoSleepS ( tstate_t newstate )
```

Puts the current thread to sleep into the specified state.

The thread goes into a sleeping state. The possible Thread States are defined into `threads.h`.

Parameters

| in  | newstate | the new thread state |
Function Class:

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

Here is the call graph for this function:

![Call Graph Image]

7.15.4.6 \texttt{chSchGoSleepTimeoutS()}

\texttt{msg_t chSchGoSleepTimeoutS (}\n\texttt{tstate_t newstate,}\n\texttt{sysinterval_t timeout })

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

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

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>newstate</th>
<th>the new thread state</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>timeout</td>
<td>the number of ticks before the operation timeouts, the special values are handled as follow:</td>
</tr>
</tbody>
</table>

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

Returns

The wakeup message.

Return values

\begin{tabular}{|c|c|}
\hline
\texttt{MSG_TIMEOUT} & if a timeout occurs. \\
\hline
\end{tabular}

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]

### 7.15.4.7 chSchWakeupS()

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

Wakes up a thread.

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

**Precondition**

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

**Note**

It is equivalent to a `chSchReadyI()` followed by a `chSchRescheduleS()` but much more efficient. The function assumes that the current thread has the highest priority.

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>in</code></td>
<td><code>ntp</code> the thread to be made ready</td>
</tr>
<tr>
<td><code>in</code></td>
<td><code>msg</code> the wakeup message</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)

### 7.15.4.8 chSchRescheduleS()

```c
void chSchRescheduleS (
    void )
```

Performs a reschedule if a higher priority thread is runnable.

If a thread with a higher priority than the current thread is in the ready list then make the higher priority thread running.

Function Class:

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

Here is the call graph for this function:
### 7.15.4.9 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>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>true</code></td>
<td>If there is a thread that must go in running state immediately.</td>
</tr>
<tr>
<td><code>false</code></td>
<td>If preemption is not required.</td>
</tr>
</tbody>
</table>

**Function Class:**

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

### 7.15.4.10 chSchDoRescheduleBehind()

```c
void chSchDoRescheduleBehind ( void )
```

Switches to the first thread on the runnable queue.

The current thread is positioned in the ready list behind all threads having the same priority. The thread regains its time quantum.

**Note**

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

**Function Class:**

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

Here is the call graph for this function:
7.15.4.11  chSchDoRescheduleAhead()

```c
void chSchDoRescheduleAhead (  
    void  )
```

Switches to the first thread on the runnable queue.

The current thread is positioned in the ready list ahead of all threads having the same priority.

**Note**

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

**Function Class:**

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

Here is the call graph for this function:

```
chSchDoRescheduleAhead  
<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ch_pqueue_remove_highest</td>
<td>chSchDoRescheduleAhead</td>
<td>ch_pqueue_insert_ahead</td>
<td>chSchReadyAhead</td>
<td>chDbgCheckClass</td>
</tr>
</tbody>
</table>
```

7.15.4.12  chSchDoReschedule()

```c
void chSchDoReschedule (  
    void  )
```

Switches to the first thread on the runnable queue.

The current thread is positioned in the ready list behind or ahead of all threads having the same priority depending on if it used its whole time slice.

**Note**

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

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

Here is the call graph for this function:

```
chSchDoReschedule
  |                |
  |                |
  v                v
ch_pqueue_remove_highest

  |                |
  |                |
  v                v
ch_pqueue_insert_ahead

  |                |
  |                |
  v                v
ch_pqueue_insert_behind
```

7.15.4.13 chSchIsRescRequiredI()

```
static bool chSchIsRescRequiredI (
    void ) [inline], [static]
```

Determines if the current thread must reschedule.

This function returns \texttt{true} if there is a ready thread with higher priority.

Returns

The priorities situation.

Return values

\begin{tabular}{|c|p{10cm}|}
\hline
false & if rescheduling is not necessary. \\
\hline
true & if there is a ready thread at higher priority. \\
\hline
\end{tabular}
Function 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:

```
chSchIsRescRequiredI chDbgCheckClassI chSysHalt
```

7.15.4.14 chSchCanYieldS()

```c
static bool chSchCanYieldS (
    void ) [inline], [static]
```

Determines if yielding is possible.

This function returns `true` if there is a ready thread with equal or higher priority.

Returns

The priorities situation.

Return values

<table>
<thead>
<tr>
<th><code>false</code></th>
<th>if yielding is not possible.</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>true</code></td>
<td>if there is a ready thread at equal or higher priority.</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:

```
chSchCanYieldS chDbgCheckClassI chSysHalt
```
### 7.15.4.15 chSchDoYieldS()

```c
static void chSchDoYieldS ( void ) [inline], [static]

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

**Function Class:**

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

Here is the call graph for this function:
```

### 7.15.4.16 chSchPreemption()

```c
static void chSchPreemption ( void ) [inline], [static]

Inline-able preemption code.
This is the common preemption code, this function must be invoked exclusively from the port layer.

**Function Class:**

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

Here is the call graph for this function:
```
7.15.5 Variable Documentation

7.15.5.1 ch

ch_system_t ch

System data structures.
7.16  Time and Intervals

7.16.1  Detailed Description

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

Special time constants

- `#define TIME_IMMEDIATE ((sysinterval_t)0)
Zero interval specification for some functions with a timeout specification.

- `#define TIME_INFINITE ((sysinterval_t)-1)
Infinite interval specification for all functions with a timeout specification.

- `#define TIME_MAX_INTERVAL ((sysinterval_t)-2)
Maximum interval constant usable as timeout.

- `#define TIME_MAX_SYSTIME ((systime_t)-1)
Maximum system of system time before it wraps.

Fast time conversion utilities

- `#define TIME_S2I(secs) ((sysinterval_t)((time_conv_t)(secs) * (time_conv_t)CH_CFG_ST_FREQUENCY))
Seconds to time interval.

- `#define TIME_MS2I(msecs)
Milliseconds to time interval.

- `#define TIME_US2I(usecs)
Microseconds to time interval.

- `#define TIME_I2S(interval)
Time interval to seconds.

- `#define TIME_I2MS(interval)
Time interval to milliseconds.

- `#define TIME_I2US(interval)
Time interval to microseconds.

Secure time conversion utilities

- static sysinterval_t chTimeS2I (time_secs_t secs)
Seconds to time interval.

- static sysinterval_t chTimeMS2I (time_msecs_t msec)
Milliseconds to time interval.

- static sysinterval_t chTimeUS2I (time_usecs_t usec)
Microseconds to time interval.

- static time_secs_t chTimeI2S (sysinterval_t interval)
Time interval to seconds.

- static time_msecs_t chTimeI2MS (sysinterval_t interval)
Time interval to milliseconds.

- static time_usecs_t chTimeI2US (sysinterval_t interval)
Time interval to microseconds.

- static systime_t chTimeAddX (systime_t systime, sysinterval_t interval)
Adds an interval to a system time returning a system time.

- static sysinterval_t chTimeDiffX (systime_t start, systime_t end)
Subtracts two system times returning an interval.

- static bool chTimeInRangeX (systime_t time, systime_t start, systime_t end)
Checks if the specified time is within the specified time range.
Typedefs

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

7.16.2 Macro Definition Documentation

7.16.2.1 TIME_IMMEDIATE

#define TIME_IMMEDIATE ((sysinterval_t)0)

Zero interval specification for some functions with a timeout specification.

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

7.16.2.2 TIME_INFINITE

#define TIME_INFINITE ((sysinterval_t)-1)

Infinite interval specification for all functions with a timeout specification.

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

7.16.2.3 TIME_MAX_INTERVAL

#define TIME_MAX_INTERVAL ((sysinterval_t)-2)

Maximum interval constant usable as timeout.
7.16.2.4 TIME_MAX_SYSTIME

#define TIME_MAX_SYSTIME ((systime_t)-1)

Maximum system of system time before it wraps.

7.16.2.5 TIME_S2I

#define TIME_S2I(secs) ((sysinterval_t)((time_conv_t)(secs) ∗ (time_conv_t)CH_CFG_ST_FREQUENCY))

Seconds to time interval.
Converts from seconds to system ticks number.

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

Parameters

| in secs | number of seconds |

Returns
The number of ticks.

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

7.16.2.6 TIME_MS2I

#define TIME_MS2I(msecs)

Value:

(sysinterval_t)(((time_conv_t)(msecs) ∗ (time_conv_t)CH_CFG_ST_FREQUENCY) + (time_conv_t)999) / (time_conv_t)1000)

Milliseconds to time interval.
Converts from milliseconds to system ticks number.

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

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>msecs</td>
</tr>
</tbody>
</table>

Returns

The number of ticks.

Function Class:

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

### 7.16.2.7 TIME_US2I

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

Value:

\[
\text{sysinterval_t}(((\text{time_conv_t})(\text{usecs}) \times \\
(\text{time_conv_t})\text{CH_CFG_ST_FREQUENCY}) + \\
(\text{time_conv_t}999999) / (\text{time_conv_t}1000000))
\]

Microseconds to time interval.

Converts from microseconds to system ticks number.

Note

The result is rounded upward to the next tick boundary.

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

Parameters

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>usecs</td>
</tr>
</tbody>
</table>

Returns

The number of ticks.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.
7.16.2.8 TIME_I2S

#define TIME_I2S(  
  interval )

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

Time interval to seconds.
Converts from system ticks number to seconds.

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

Parameters
in 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.

7.16.2.9 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.

7.16.2.10 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.

7.16.3 Typedef Documentation
7.16 Time and Intervals

7.16.3.1 systime_t

typedef uint64_t systime_t

Type of system time.

Note

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

7.16.3.2 sysinterval_t

typedef uint64_t sysinterval_t

Type of time interval.

Note

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

7.16.3.3 time_secs_t

typedef uint32_t time_secs_t

Type of seconds.

Note

It is selectable in configuration between 16 or 32 bits.

7.16.3.4 time_msecs_t

typedef uint32_t time_msecs_t

Type of milliseconds.

Note

It is selectable in configuration between 16 or 32 bits.
7.16.3.5 time_usecs_t

typedef uint32_t time_usecs_t

Type of microseconds.

Note
It is selectable in configuration between 16 or 32 bits.

7.16.3.6 time_conv_t

typedef uint64_t time_conv_t

Type of time conversion variable.

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

7.16.4 Function Documentation

7.16.4.1 chTimeS2I()

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

Seconds to time interval.

Converts from seconds to system ticks number.

Note
The result is rounded upward to the next tick boundary.

Parameters

| in  | secs | number of seconds |

Returns

The number of ticks.
Function Class:
   Special function, this function has special requirements see the notes.

### 7.16.4.2 chTimeMS2I()

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

Milliseconds to time interval.

Converts from milliseconds to system ticks number.

**Note**
   The result is rounded upward to the next tick boundary.

**Parameters**
   
   ```
   in   msec  number of milliseconds
   ```

**Returns**
   The number of ticks.

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

### 7.16.4.3 chTimeUS2I()

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

Microseconds to time interval.

Converts from microseconds to system ticks number.

**Note**
   The result is rounded upward to the next tick boundary.

**Parameters**
   
   ```
   in   usec  number of microseconds
   ```

ChibiOS/RT
Returns

The number of ticks.

Function Class:

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

7.16.4.4 chTimeI2S()

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

Time interval to seconds.

Converts from system interval to seconds.

Note

The result is rounded up to the next second boundary.

Parameters

| in | interval | interval in ticks |

Returns

The number of seconds.

Function Class:

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

7.16.4.5 chTimeI2MS()

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

Time interval to milliseconds.

Converts from system interval to milliseconds.

Note

The result is rounded up to the next millisecond boundary.
7.16.4.6 chTimeI2US()

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

Time interval to microseconds.
Converts from system interval to microseconds.

Note
The result is rounded up to the next microsecond boundary.

Parameters

in interval interval in ticks

Returns
The number of microseconds.

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

7.16.4.7 chTimeAddX()

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

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

<table>
<thead>
<tr>
<th>in</th>
<th>systime</th>
<th>base system time</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>interval</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.

7.16.4.8 chTimeDiffX()

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

Subtracts two system times returning an interval.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>start</th>
<th>first system time</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>end</td>
<td>second system time</td>
</tr>
</tbody>
</table>

Returns

The interval representing the time difference.

Function Class:

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

7.16.4.9 chTimeIsInRangeX()

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

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

Note

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

<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

| 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.
7.17 Virtual Timers

7.17.1 Detailed Description

Time and Virtual Timers related APIs and services.

Functions

- static void vt_init (delta_list_t *dlhp)
  
  Delta list initialization.

- static bool vt_is_empty (delta_list_t *dlhp)
  
  List empty check.

- static bool vt_is_last (delta_list_t *dlhp, delta_list_t *dlp)
  
  Last timer in the list check.

- static bool vt_is_first (delta_list_t *dlhp, delta_list_t *dlp)
  
  First timer in the list check.

- static bool vt_is_timer (delta_list_t *dlhp, delta_list_t *dlp)
  
  Timer check.

- static void vt_insert_after (delta_list_t *dlhp, delta_list_t *dlp, sysinterval_t delta)
  
  Inserts an element after another header element.

- static void vt_insert_before (delta_list_t *dlhp, delta_list_t *dlp, sysinterval_t delta)
  
  Inserts an element before another header element.

- static void vt_insert (delta_list_t *dlhp, delta_list_t *dlep, sysinterval_t delta)
  
  Inserts an element in a delta list.

- static delta_list_t * vt_remove_first (delta_list_t *dlhp)
  
  Dequeues an element from the delta list.

- static delta_list_t * vt_dequeue (delta_list_t *dlp)
  
  Dequeues an element from the delta list.

- static void vt_set_alarm (systime_t now, sysinterval_t delay)
  
  Alarm time setup.

- static void vt_insert_first (virtual_timers_list_t *vtlp, virtual_timer_t *vtp)
  
  Inserts a timer as first element in a delta list.

- static void vt_enqueue (virtual_timers_list_t *vtlp, virtual_timer_t *vtp, sysinterval_t delay)
  
  Enqueues a virtual timer in a virtual timers list.

- void _vt_init (void)
  
  Virtual Timers initialization.

- void chVTDosetI (virtual_timer_t *vtp, sysinterval_t delay, vtfunc_t vtfunc, void *par)
  
  Enables a one-shot virtual timer.

- void chVTDoresetI (virtual_timer_t *vtp)
  
  Disables a Virtual Timer.

- sysinterval_t chVTGetRemainingIntervalI (virtual_timer_t *vtp)
  
  Returns the remaining time interval before next timer trigger.

- void chVTDoTickI (void)
  
  Virtual timers ticker.

- static void chVTObjectinit (virtual_timer_t *vtp)
  
  Initializes a virtual_timer_t object.

- static systime_t chVTGetSystemTimeX (void)
  
  Current system time.

- static systime_t chVTGetSystemTime (void)
### 7.17 Virtual Timers

Current system time:

- **static sysinterval_t chVTTimeElapsedSinceX (systime_t start)**
  
  Returns the elapsed time since the specified start time.

- **static bool chVTIsSystemTimeWithinX (systime_t start, systime_t end)**
  
  Checks if the current system time is within the specified time window.

- **static bool chVTIsSystemTimeWithin (systime_t start, systime_t end)**
  
  Checks if the current system time is within the specified time window.

- **static bool chVTGetTimersStateI (sysinterval_t *timep)**
  
  Returns the time interval until the next timer event.

- **static bool chVTIsArmedI (const virtual_timer_t *vtp)**
  
  Returns true if the specified timer is armed.

- **static bool chVTIsArmed (const virtual_timer_t *vtp)**
  
  Returns true if the specified timer is armed.

- **static void chVTResetI (virtual_timer_t *vtp)**
  
  Disables a Virtual Timer.

- **static void chVTReset (virtual_timer_t *vtp)**
  
  Disables a Virtual Timer.

- **static void chVTSetI (virtual_timer_t *vtp, sysinterval_t delay, vtfunc_t vtfunc, void *par)**
  
  Enables a virtual timer.

- **static void chVTSet (virtual_timer_t *vtp, sysinterval_t delay, vtfunc_t vtfunc, void *par)**
  
  Enables a virtual timer.

#### 7.17.2 Function Documentation

##### 7.17.2.1 vt_init()

```c
static void vt_init (  
    delta_list_t * dlhp );  [inline], [static]
```

Delta list initialization.

**Parameters**

| out dlhp | pointer to the delta list header |

**Function Class:**

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

##### 7.17.2.2 vt_is_empty()

```c
static bool vt_is_empty (  
    delta_list_t * dlhp );  [inline], [static]
```

List empty check.
Parameters

| in  | dlhp | pointer to the delta list header |

Function Class:

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

### 7.17.2.3 vt_is_last()

```c
static bool vt_is_last(
    delta_list_t * dlhp,
    delta_list_t * dlp)
```

Last timer in the list check.

Parameters

| in  | dlhp | pointer to the delta list header |
| in  | dlp  | pointer to the delta list element |

Function Class:

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

### 7.17.2.4 vt_is_first()

```c
static bool vt_is_first(
    delta_list_t * dlhp,
    delta_list_t * dlp)
```

First timer in the list check.

Parameters

| in  | dlhp | pointer to the delta list header |
| in  | dlp  | pointer to the delta list element |

Function Class:

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

7.17.2.5 vt_is_timer()

```c
static bool vt_is_timer (  
    delta_list_t * dlhp,  
    delta_list_t * dlp )  [inline], [static]
```
Timer check.

**Parameters**

| in | dlhp | pointer to the delta list header |
| in | dlp  | pointer to the delta list element |

**Function Class:**

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

7.17.2.6 vt_insert_after()

```c
static void vt_insert_after (  
    delta_list_t * dlhp,  
    delta_list_t * dlp,  
    sysinterval_t delta )  [inline], [static]
```
Inserts an element after another header element.

**Parameters**

| in | dlhp | pointer to the delta list header element |
| in | dlp  | element to be inserted after the header element |
| in | delta | delta of the element to be inserted |

**Function Class:**

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

7.17.2.7 vt_insert_before()

```c
static void vt_insert_before (  
    delta_list_t * dlhp,  
    delta_list_t * dlp,  
    sysinterval_t delta )  [inline], [static]
```
Inserts an element before another header element.
Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>dlhp</th>
<th>pointer to the delta list header element</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>dlep</td>
<td>element to be inserted before the header element</td>
</tr>
<tr>
<td>in</td>
<td>delta</td>
<td>delta of the element to be inserted</td>
</tr>
</tbody>
</table>

Function Class:

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

7.17.2.8 vt_insert()

```c
static void vt_insert (  
    delta_list_t * dlhp,  
    delta_list_t * dlep,  
    sysinterval_t delta ) [inline], [static]
```

Inserts an element in a delta list.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>dlhp</th>
<th>pointer to the delta list header element</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>dlep</td>
<td>element to be inserted before the header element</td>
</tr>
<tr>
<td>in</td>
<td>delta</td>
<td>delta of the element to be inserted</td>
</tr>
</tbody>
</table>

Function Class:

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

Here is the call graph for this function:

```
vt_insert  vt_insert_before
```

7.17.2.9 vt_remove_first()

```c
static delta_list_t* vt_remove_first (  
    delta_list_t * dlhp ) [inline], [static]
```

Dequeues an element from the delta list.
7.17 Virtual Timers

Parameters

| in  | dlhp  | pointer to the delta list header |

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

7.17.2.10 vt_dequeue()

static delta_list_t* vt_dequeue (delta_list_t * dlp) [inline], [static]

Dequeues an element from the delta list.

Parameters

| in  | dlp  | pointer to the delta list element |

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

7.17.2.11 vt_set_alarm()

static void vt_set_alarm (systime_t now, sysinterval_t delay) [static]

Alarm time setup.

Note
An RFCU fault is registered if the system time skips past (now + delay), the deadline is skipped forward in order to compensate for the event.

Parameters

| in  | now  | last known system time |
| in  | delay | delay over now |
Here is the call graph for this function:

![Call Graph for vt_set_alarm](image)

### 7.17.2.12 vt_insert_first()

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

Inserts a timer as first element in a delta list.

**Note**

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

Here is the call graph for this function:

![Call Graph for vt_insert_first](image)
### 7.17.2.13 vt_enqueue()

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

Enqueues a virtual timer in a virtual timers list.

Here is the call graph for this function:

![Call Graph](image)

### 7.17.2.14 _vt_init()

```c
void _vt_init (
    void )
```

Virtual Timers initialization.

**Note**

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

Here is the call graph for this function:

```
 _vt_init   vt_init
```

7.17.2.15 chVTDoSetI()

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

Enables a one-shot virtual timer.

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

Precondition

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

Note

The callback function is invoked from interrupt context.

Parameters

<table>
<thead>
<tr>
<th>out</th>
<th>vtp</th>
<th>the virtual_timer_t structure pointer</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>delay</td>
<td>the number of ticks before the operation timeouts, the special values are handled as follow:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• TIME_INFINITE is allowed but interpreted as a normal time specification.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• TIME_IMMEDIATE this value is not allowed.</td>
</tr>
<tr>
<td>in</td>
<td>vtfnc</td>
<td>the timer callback function. After invoking the callback the timer is disabled and the structure can be disposed or reused.</td>
</tr>
<tr>
<td>in</td>
<td>par</td>
<td>a parameter that will be passed to the callback function</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:

```
7.17.2.16 chVTDoResetI()

void chVTDoResetI (  
    virtual_timer_t * vtp  )
```

Disables a Virtual Timer.

**Precondition**

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

**Parameters**

```
in  vtp  the virtual_timer_t structure pointer  
```
Function Class:

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

Here is the call graph for this function:

```
7.17.2.17 chVTGetRemainingIntervalI()

sysinterval_t chVTGetRemainingIntervalI (virtual_timer_t * vtp )

Returns the remaining time interval before next timer trigger.

Note

This function can be called while the timer is active.

Parameters

| in  | vtp | the virtual_timer_t structure pointer |

Returns

The remaining time interval.
Function Class:

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

Here is the call graph for this function:

7.17.2.18 chVTDoTickI()

```c
void chVTDoTickI (  
    void  )
```

Virtual timers ticker.

Note

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

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

Here is the call graph for this function:

![Call Graph](image-url)

### 7.17.2.19 chVObjectInit()

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

**Initializes a virtual_timer_t object.**

**Note**

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

**Parameters**

| out | vtp | the virtual_timer_t structure pointer |
Function Class:

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

### 7.17.2.20 chVTGetSystemTimeX()

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

Current system time.

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

**Note**

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

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

**Returns**

The system time in ticks.

Function Class:

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

### 7.17.2.21 chVTGetSystemTime()

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

Current system time.

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

**Note**

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

**Returns**

The system time in ticks.
Function Class:

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

Here is the call graph for this function:

7.17.2.22  chVTTimeElapsedSinceX()

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

Returns the elapsed time since the specified start time.

Parameters

| in | start  | start time |

Returns

The elapsed time.
Function Class:

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

Here is the call graph for this function:

```
chVTIsSystemTimeWithinX()  
^                              ^
|                              |  ch VT Get System Time X
|                              |  ch Time Diff X
|                              |  ch VT Time Elapsed Since X
```

### 7.17.2.23 chVTIsSystemTimeWithinX()

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

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

**Note**

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

**Parameters**

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

**Return values**

<table>
<thead>
<tr>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>true</strong></td>
<td></td>
<td>current time within the specified time window.</td>
</tr>
<tr>
<td><strong>false</strong></td>
<td></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.

Here is the call graph for this function:

```
chVTIsSystemTimeWithinX
  ↓
  |    ↓
  |   chTimeIsInRangeX
  |    ↓
  |   chVTGetSystemTimeX
```

---

### 7.17.2.24 `chVTIsSystemTimeWithin()`

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

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

**Note**

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

**Parameters**

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

**Return values**

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

```
chVTIsSystemTimeWithin
chTimeIsInRangeX
chVTGetSystemTime
chSysLock
chSysUnlock
chVTGetSystemTimeX
```

### 7.17.2.25 chVTGetTimersStateI()

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

Returns the time interval until the next timer event.

**Note**

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

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

**Parameters**

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

**Returns**

The time, in ticks, until next time event.

**Return values**

- `false` if the timers list is empty.
- `true` if the timers list contains at least one timer.
Function Class:

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

Here is the call graph for this function:

![Call graph](image)

### 7.17.2.26 chVTIsArmedI()

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

**Returns** true if the specified timer is armed.

**Precondition**

The timer must have been initialized using `chVTOBJECT_Init()` or `chVTDOSetI()`.

**Parameters**

| in  | vtp | the virtual_timer_t structure pointer |

**Returns**

true if the timer is armed.
Function Class:

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

Here is the call graph for this function:

```
chVTIsArmedI ─── chDbgCheckClassI ─── chSysHalt
```

### 7.17.2.27 chVTIsArmed()

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

Returns **true** if the specified timer is armed.

**Precondition**

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

**Parameters**

- **in** `vtp` the `virtual_timer_t structure pointer`

Returns

**true** if the timer is armed.
Function Class:

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

Here is the call graph for this function:

```
chVTIsArmed
chSysLock
chSysUnlock
chVTIsArmedI
_dbg_check_lock
_stats_start_measure
_crit_thd
_dbg_check_unlock
_stats_stop_measure
_crit_thd
chDbgCheckClassI
```

### 7.17.2.28 chVTResetI()

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

Disables a Virtual Timer.

**Note**

The timer is first checked and disabled only if armed.

**Precondition**

The timer must have been initialized using `chVTOBJECTInit()` or `chVTDoSetI()`.

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>in</th>
<th>vtp</th>
<th>the virtual_timer_t structure pointer</th>
</tr>
</thead>
</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_url)

### 7.17.2.29 chVTReset()

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

Disables a Virtual Timer.

**Note**

The timer is first checked and disabled only if armed.

**Precondition**

The timer must have been initialized using `chVTOBJECTInit()` or `chVTDOSetI()`.

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>in</code> vtp</td>
<td>the <code>virtual_timer_t</code> structure pointer</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:

```
chVTReset
chSysLock
chSysUnlock
chVTResetI
_dbg_check_lock
_stats_start_measure
_crit_thd
_dbg_check_unlock
_stats_stop_measure
_crit_thd
chVTDoResetI
chVTIsArmed!
```

### 7.17.2.30 chVTSetI()

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

Enables a virtual timer.

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

**Precondition**

The timer must have been initialized using `chVTOBJECTInit()` or `chVTDoSetI()`.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in vtp</td>
<td>virtual_timer_t structure pointer</td>
<td>the virtual_timer_t structure pointer</td>
</tr>
<tr>
<td>in delay</td>
<td>sysinterval_t</td>
<td>the number of ticks before the operation timeouts, the special values are handled as follow:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• TIME_INFINITE is allowed but interpreted as a normal time specification.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• TIME_IMMEDIATE this value is not allowed.</td>
</tr>
<tr>
<td>in vtfunc</td>
<td>vtfunc_t</td>
<td>the timer callback function. After invoking the callback the timer is disabled and the structure can be disposed or reused.</td>
</tr>
<tr>
<td>in par</td>
<td>void</td>
<td>a parameter that will be passed to the callback function</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:

```
chVTSet() -> chVTDoSetI() -> chVTResetI() 
                      ^------------------------^    ^------------------------^    ^------------------------^ 
                      |                        |    |                        |    |                        | 
                      |                        |    |                        |    |                        | 
                      v                        v    v                        v    v                        v 
vt_enqueue() -> chDBGCheckClass() 
```

### 7.17.2.31 chVTSet()

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

Enables a virtual timer.

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

**Precondition**

The timer must have been initialized using `chVObjectInit()` or `chVTDoSetI()`.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>vtp</strong></td>
<td>the virtual_timer_t structure pointer</td>
</tr>
<tr>
<td><strong>delay</strong></td>
<td>the number of ticks before the operation timeouts, the special values are handled as follow:</td>
</tr>
<tr>
<td></td>
<td>- TIME_INFINITE is allowed but interpreted as a normal time specification.</td>
</tr>
<tr>
<td></td>
<td>- TIME_IMMEDIATE this value is not allowed.</td>
</tr>
<tr>
<td><strong>vtfnc</strong></td>
<td>the timer callback function. After invoking the callback the timer is disabled and the structure can be disposed or reused.</td>
</tr>
<tr>
<td><strong>par</strong></td>
<td>a parameter that will be passed to the callback function</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:
7.18 Threads

7.18.1 Detailed Description

Threads related APIs and services.

Operation mode

A thread is an abstraction of an independent instructions flow. In ChibiOS/RT a thread is represented by a "C" function owning a processor context, state informations and a dedicated stack area. In this scenario static variables are shared among all threads while automatic variables are local to the thread.

Operations defined for threads:

- **Create**, a thread is started on the specified thread function. This operation is available in multiple variants, both static and dynamic.

- **Exit**, a thread terminates by returning from its top level function or invoking a specific API, the thread can return a value that can be retrieved by other threads.

- **Wait**, a thread waits for the termination of another thread and retrieves its return value.

- **Resume**, a thread created in suspended state is started.

- **Sleep**, the execution of a thread is suspended for the specified amount of time or the specified future absolute time is reached.

- **SetPriority**, a thread changes its own priority level.

- **Yield**, a thread voluntarily renounces to its time slot.

Threads queues

- `#define _THREADS_QUEUE_DATA(name) {__CH_QUEUE_DATA(name)}`
  
  Data part of a static threads queue object initializer.

- `#define THREADS_QUEUE_DECL(name) threads_queue_t name = _THREADS_QUEUE_DATA(name)`
  
  Static threads queue object initializer.

Working Areas

- `#define THD_WORKING_AREA_SIZE(n) MEM_ALIGN_NEXT(sizeof(thread_t) + PORT_WA_SIZE(n), PORT_STACK_ALIGN)`
  
  Calculates the total Working Area size.

- `#define THD_WORKING_AREA(s, n) PORT_WORKING_AREA(s, n)`
  
  Static working area allocation.

- `#define THD_WORKING_AREA_BASE(s) ((stkalign_t *)(s))`
  
  Base of a working area casted to the correct type.

- `#define THD_WORKING_AREA_END(s)`
  
  End of a working area casted to the correct type.
Threads abstraction macros

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

Macro Functions

• 
  #define chThdSleepSeconds(sec) chThdSleep(TIME_S2I(sec))
  
  Delays the invoking thread for the specified number of seconds.

• 
  #define chThdSleepMilliseconds(msec) chThdSleep(TIME_MS2I(msec))
  
  Delays the invoking thread for the specified number of milliseconds.

• 
  #define chThdSleepMicroseconds(usec) chThdSleep(TIME_US2I(usec))
  
  Delays the invoking thread for the specified number of microseconds.

Typedefs

• 
  typedef void(*tfunc_t)(void*p)
  
  Thread function.

Data Structures

• 
  struct thread_descriptor_t
  
  Type of a thread descriptor.

Functions

• 
  thread_t*_thread_init(thread_t*tp, const char*name, tprio_tprio)
  
  Initializes a thread structure.

• 
  void_thread_memfill(uint8_t*startp, uint8_t*endp, uint8_t v)
  
  Memory fill utility.

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

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

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

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

• 
  thread_t*chThdCreateStatic(void*wsp, size_tsize, tprio_tprio, tfunc_tpf, void*arg)
  
  Creates a new thread into a static memory area.

• 
  thread_t*chThdStart(thread_t*tp)
  
  Resumes a thread created with chThdCreateI().

• 
  thread_t*chThdAddRef(thread_t*tp)
  
  Adds a reference to a thread object.

• 
  voidchThdRelease(thread_t*tp)
  
  Releases a reference to a thread object.

• 
  voidchThdExit(msg_tmsg)
  
  Terminates the current thread.
• void chThdExitS (msg_t msg)
  Terminates the current thread.
• msg_t chThdWait (thread_t *tp)
  Blocks the execution of the invoking thread until the specified thread terminates then the exit code is returned.
• tprio_t chThdSetPriority (tprio_t newprio)
  Changes the running thread priority level then reschedules if necessary.
• void chThdTerminate (thread_t *tp)
  Requests a thread termination.
• void chThdSleep (sysinterval_t time)
  Suspends the invoking thread for the specified time.
• void chThdSleepUntil (systime_t time)
  Suspends the invoking thread until the system time arrives to the specified value.
• systime_t chThdSleepUntilWindowed (systime_t prev, systime_t next)
  Suspends the invoking thread until the system time arrives to the specified value.
• void chThdYield (void)
  Yields the time slot.
• msg_t chThdSuspendS (thread_reference_t *trp)
  Sends the current thread sleeping and sets a reference variable.
• msg_t chThdSuspendTimeoutS (thread_reference_t *trp, sysinterval_t timeout)
  Sends the current thread sleeping and sets a reference variable.
• void chThdResumeI (thread_reference_t *trp, msg_t msg)
  Wakes up a thread waiting on a thread reference object.
• void chThdResumeS (thread_reference_t *trp, msg_t msg)
  Wakes up a thread waiting on a thread reference object.
• void chThdResume (thread_reference_t *trp, msg_t msg)
  Wakes up a thread waiting on a thread reference object.
• msg_t chThdEnqueueTimeoutS (threads_queue_t *tqp, sysinterval_t timeout)
  Enqueues the caller thread on a threads queue object.
• void chThdDequeueNextI (threads_queue_t *tqp, msg_t msg)
  Dequeues and wakes up one thread from the threads queue object, if any.
• void chThdDequeueAllI (threads_queue_t *tqp, msg_t msg)
  Dequeues and wakes up all threads from the threads queue object.
• static thread_t * chThdGetSelfX (void)
  Returns a pointer to the current thread_t.
• static tprio_t chThdGetPriorityX (void)
  Returns the current thread priority.
• static systime_t chThdGetTicksX (thread_t *tp)
  Returns the number of ticks consumed by the specified thread.
• static stkalign_t * chThdGetWorkingAreaX (thread_t *tp)
  Returns the working area base of the specified thread.
• static bool chThdTerminatedX (thread_t *tp)
  Verifies if the specified thread is in the CH_STATE_FINAL state.
• static bool chThdShouldTerminateX (void)
  Verifies if the current thread has a termination request pending.
• static thread_t * chThdStartI (thread_t *tp)
  Resumes a thread created with chThdCreateI().
• static void chThdSleepS (sysinterval_t ticks)
  Suspends the invoking thread for the specified number of ticks.
• static void chThdQueueObjectInit (threads_queue_t *tqp)
  Initializes a threads queue object.
• static bool chThdQueuesEmptyI (threads_queue_t *tqp)
  Evaluates to true if the specified queue is empty.
• static void chThdDoDequeueNextI (threads_queue_t *tqp, msg_t msg)
  Dequeues and wakes up one thread from the threads queue object.
7.18.2 Macro Definition Documentation

7.18.2.1 _THREADS_QUEUE_DATA

#define _THREADS_QUEUE_DATA( name ) _CH_QUEUE_DATA(name)

Data part of a static threads queue object initializer.

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

Parameters

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

7.18.2.2 THREADS_QUEUE_DECLS

#define THREADS_QUEUE_DECLS( 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 |

7.18.2.3 THD_WORKING_AREA_SIZE

#define THD_WORKING_AREA_SIZE( n ) MEM_ALIGN_NEXT(sizeof(thread_t) + PORT_WA_SIZE(n), PORT_STACK_ALIGN)

Calculates the total Working Area size.

Parameters

| in | n | the stack size to be assigned to the thread |
7.18 Threads

Returns

The total used memory in bytes.

Function Class:

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

7.18.2.4 THD_WORKING_AREA

#define THD_WORKING_AREA(  
    s,
    n ) PORT_WORKING_AREA(s, n)

Static working area allocation.

This macro is used to allocate a static thread working area aligned as both position and size.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>s</th>
<th>the name to be assigned to the stack array</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>n</td>
<td>the stack size to be assigned to the thread</td>
</tr>
</tbody>
</table>

Function Class:

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

7.18.2.5 THD_WORKING_AREA_BASE

#define THD_WORKING_AREA_BASE(  
    s ) ((stkalign_t *) (s))

Base of a working area casted to the correct type.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>s</th>
<th>name of the working area</th>
</tr>
</thead>
</table>

7.18.2.6 THD_WORKING_AREA_END

#define THD_WORKING_AREA_END(  
    s )
Value:

End of a working area casted to the correct type.

Parameters

| in | s   | name of the working area |

### 7.18.2.7 THD_FUNCTION

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

Thread declaration macro.

**Note**

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

### 7.18.2.8 chThdSleepSeconds

```c
#define chThdSleepSeconds(sec) chThdSleep(TIME_S2I(sec))
```

Delays the invoking thread for the specified number of seconds.

**Note**

The specified time is rounded up to a value allowed by the real system tick clock.
The maximum specifiable value is implementation dependent.
Use of this macro for large values is not secure because integer overflows, make sure your value can be correctly converted.

Parameters

| in | sec | time in seconds, must be different from zero |

Function Class:

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

```c
#define chThdSleepMilliseconds(msec) chThdSleep(TIME_MS2I(msec))
```

Delays the invoking thread for the specified number of milliseconds.

**Note**

The specified time is rounded up to a value allowed by the real system tick clock.

The maximum specifiable value is implementation dependent.

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

**Parameters**

| in  | msec | time in milliseconds, must be different from zero |

**Function Class:**

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

7.18.2.10 chThdSleepMicroseconds

```c
#define chThdSleepMicroseconds(usec) chThdSleep(TIME_US2I(usec))
```

Delays the invoking thread for the specified number of microseconds.

**Note**

The specified time is rounded up to a value allowed by the real system tick clock.

The maximum specifiable value is implementation dependent.

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

**Parameters**

| in  | usec | time in microseconds, must be different from zero |

**Function Class:**

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

7.18.3 Typedef Documentation
### 7.18.3.1 tfunc_t

```c
typedef void(* tfunc_t) (void *)
```

Thread function.

### 7.18.4 Function Documentation

#### 7.18.4.1 _thread_init()

```c
define thread_t * _thread_init ( 
    thread_t * tp,
    const char * name,
    tprio_t prio )
```

Initializes a thread structure.

**Note**

This is an internal functions, do not use it in application code.

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>tp</td>
<td>pointer to the thread</td>
</tr>
<tr>
<td>in</td>
<td>name</td>
<td>thread name</td>
</tr>
<tr>
<td>in</td>
<td>prio</td>
<td>the priority level for the new thread</td>
</tr>
</tbody>
</table>

**Returns**

The same thread pointer passed as parameter.
Function Class:
Not an API, this function is for internal use only.

Here is the call graph for this function:

```
ch_list_init
_thread_init
ch_queue_init
chTMObjectInit
```

### 7.18.4.2 _thread_memfill()

```c
void _thread_memfill (
    uint8_t * startp,
    uint8_t * endp,
    uint8_t v )
```

Memory fill utility.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>startp</th>
<th>first address to fill</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>endp</td>
<td>last address to fill +1</td>
</tr>
<tr>
<td>in</td>
<td>v</td>
<td>filler value</td>
</tr>
</tbody>
</table>

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

### 7.18.4.3 chThdCreateSuspendedI()

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

Creates a new thread into a static memory area.

The new thread is initialized but not inserted in the ready list, the initial state is `CH_STATE_WTSTART`. 
Postcondition

The created thread has a reference counter set to one, it is caller responsibility to call \texttt{chThdRelease()} or \texttt{chthdWait()} in order to release the reference. The thread persists in the registry until its reference counter reaches zero.

The initialized thread can be subsequently started by invoking \texttt{chThdStart()}, \texttt{chThdStartI()} or \texttt{chSchWakeupS()} depending on the execution context.

Note

A thread can terminate by calling \texttt{chThdExit()} or by simply returning from its main function.

Threads created using this function do not obey to the \texttt{CH_DBG_FILL_THREADS} debug option because it would keep the kernel locked for too much time.

Parameters

\begin{verbatim}
  out tdp pointer to the thread descriptor
\end{verbatim}

Returns

The pointer to the \texttt{thread_t} structure allocated for the thread into the working space area.

Function Class:

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

Here is the call graph for this function:

```
ch_thrdCreateSuspendedI
  \_thread_init
  ch dbgCheckClassI
  ch list_init
  ch queue_init
  ch TMobjecInit
  ch sysHalt
```

ChibiOS/RT
7.18 Threads

7.18.4.4  chThdCreateSuspended()

thread_t * chThdCreateSuspended (  
    const thread_descriptor_t * tdp )

Creates a new thread into a static memory area.

The new thread is initialized but not inserted in the ready list, the initial state is CH_STATE_WTSSTART.

Postcondition

The created thread has a reference counter set to one, it is caller responsibility to call chThdRelease() or chthdWait() in order to release the reference. The thread persists in the registry until its reference counter reaches zero.

The initialized thread can be subsequently started by invoking chThdStart(), chThdStartI() or chSchWakeupS() depending on the execution context.

Note

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

Parameters

| out    | tdp | pointer to the thread descriptor |

Returns

The pointer to the thread_t structure allocated for the thread into the working space area.

Function Class:

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

Here is the call graph for this function:
7.18.4.5 chThdCreateI()

thread_t * chThdCreateI (  
    const thread_descriptor_t * tdp )

Creates a new thread into a static memory area.

The new thread is initialized and make ready to execute.

Postcondition

The created thread has a reference counter set to one, it is caller responsibility to call chThdRelease() or chthdWait() in order to release the reference. The thread persists in the registry until its reference counter reaches zero.

The initialized thread can be subsequently started by invoking chThdStart(), chThdStartI() or chSchWakeupS() depending on the execution context.

Note

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

Threads created using this function do not obey to the CH_DBG_FILL_THREADS debug option because it would keep the kernel locked for too much time.

Parameters

| out | tdp | pointer to the thread descriptor |

Returns

The pointer to the thread_t structure allocated for the thread into the working space area.

Function Class:

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

Here is the call graph for this function:
7.18.4.6  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.

**Postcondition**

The created thread has a reference counter set to one, it is caller responsibility to call `chThdRelease()` or `chthdWait()` in order to release the reference. The thread persists in the registry until its reference counter reaches zero.

**Note**

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

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>out</code></td>
<td><code>tdp</code></td>
<td>pointer to the thread descriptor</td>
</tr>
</tbody>
</table>

**Returns**

The pointer to the `thread_t` structure allocated for the thread into the working space area.

**Function Class:**

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

![Call Graph](image_url)

### 7.18.4.7 chThdCreateStatic()

```c
thread_t * chThdCreateStatic (  
  void * wsp,  
  size_t size,  
  tprio_t prio,  
  tfunc_t pf,  
  void * arg )
```

Creates a new thread into a static memory area.

**Postcondition**

The created thread has a reference counter set to one, it is caller responsibility to call `chThdRelease()` or `chThdWait()` in order to release the reference. The thread persists in the registry until its reference counter reaches zero.

**Note**

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

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>out</code></td>
<td><code>wsp</code> pointer to a working area dedicated to the thread stack</td>
</tr>
<tr>
<td><code>in</code></td>
<td><code>size</code> size of the working area</td>
</tr>
<tr>
<td><code>in</code></td>
<td><code>prio</code> the priority level for the new thread</td>
</tr>
<tr>
<td><code>in</code></td>
<td><code>pf</code> the thread function</td>
</tr>
<tr>
<td><code>in</code></td>
<td><code>arg</code> an argument passed to the thread function. It can be <code>NULL</code>.</td>
</tr>
</tbody>
</table>
Returns

The pointer to the thread_t structure allocated for the thread into the working space area.

Function Class:

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

Here is the call graph for this function:

7.18.4.8 chThdStart()

thread_t * chThdStart ( thread_t * tp )

Resumes a thread created with chThdCreateI().

Parameters

| in | tp | pointer to the thread |

Returns

The pointer to the thread_t structure allocated for the thread into the working space area.
Function Class:

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

Here is the call graph for this function:

```
chThdStart
 chSchWakeupS
  
  chSysLock
   
   chSysUnlock
    
    chDbgCheckClassS
      chSchReadyAheadI
        chSchReadyI
          _dbg_check_lock
            _stats_start_measure
              _crit_thd
                _dbg_check_unlock
                  _stats_stop_measure
                    _crit_thd
```

7.18.4.9 chThdAddRef()

```c
thread_t * chThdAddRef ( 
  thread_t * tp )
```

Adds a reference to a thread object.

Precondition

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

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>tp</th>
<th>pointer to the thread</th>
</tr>
</thead>
</table>
Returns

The same thread pointer passed as parameter representing the new reference.

Function Class:

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

Here is the call graph for this function:

```
chThdAddRef
  |          |          |
  v          v          v
chSysLock    _dbg_check_lock
  |          |          |
  v          v          v
_stats_start_measure
  |          |          |
  v          v          v
_crit_thd  _dbg_check_unlock
  |          |          |
  v          v          v
_stats_stop_measure
  |          |          |
  v          v          v
_crit_thd
```

7.18.4.10 chThdRelease()

```c
void chThdRelease (
    thread_t * tp )
```

Releases a reference to a thread object.

If the references counter reaches zero and the thread is in the CH_STATE_FINAL state then the thread’s memory is returned to the proper allocator and the thread is removed from the registry. Threads whose counter reaches zero and are still active become “detached” and will be removed from registry on termination.

Precondition

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

Note

Static threads are not affected.
Parameters

| in  | tp  | pointer to the thread |

Function Class:

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

Here is the call graph for this function:

```
chThdRelease
    |         |
    |         |
chHeapFree  chSysLock  chPoolFreeI
    |         |         |
    |         |         |
chThdGetWorkingAreaX      chPoolFree
    |         |
    |         |
chSysUnlock
```

7.18.4.11 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

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

Parameters

| in | msg | thread exit code |
Function Class:

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

Here is the call graph for this function:

![Call Graph Diagram]

7.18.4.12 chThdExitS()

```c
void chThdExitS (msg_t msg)
```

Terminates the current thread.

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

Postcondition

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

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

Parameters

| in | msg | thread exit code |

ChibiOS/RT
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:

```
ch_list_notempty
    chThdExitS
        ch_list_pop
    chSchGoSleepS
        chDbgCheckClassS
    chSchReadyI
        ch_pqueue_remove_highest
        ch_pqueue_insert_behind
        chDbgCheckClassI
```

### 7.18.4.13 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.

This function waits for the specified thread to terminate then decrements its reference counter, if the counter reaches zero then the thread working area is returned to the proper allocator and the thread is removed from registry.

**Precondition**

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

**Postcondition**

Enabling **chThdWait()** requires 2-4 (depending on the architecture) extra bytes in the **thread_t** structure.

**Note**

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

| in  | tp  | pointer to the thread |

Returns

The exit code from the terminated thread.

Function Class:

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

Here is the call graph for this function:

![Call Graph](image)

7.18.4.14 chThdSetPriority()

tprio_t chThdSetPriority (
    tprio_t newprio
)

Changes the running thread priority level then reschedules if necessary.

Note

The function returns the real thread priority regardless of the current priority that could be higher than the real priority because the priority inheritance mechanism.

Parameters

| in  | newprio | the new priority level of the running thread |
Returns

The old priority level.

Function Class:

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

Here is the call graph for this function:

![Call Graph]

---

### 7.18.4.15 chThdTerminate()

```c
void chThdTerminate (
    thread_t * tp )
```

Requests a thread termination.

Precondition

The target thread must be written to invoke periodically `chThdShouldTerminate()` and terminate cleanly if it returns `true`.

Postcondition

The specified thread will terminate after detecting the termination condition.
Parameters

```c
in   tp    pointer to the thread
```

Function Class:

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

Here is the call graph for this function:

```
chThdTerminate
        |   \    ___
        |       |     _dbg_check_lock
        |       |     __stats_start_measure
        |       |     ___crit_thd
        |         \    ___
        |           |     _dbg_check_unlock
        |           |     __stats_stop_measure
        |           |     ___crit_thd
        \         \     chSysUnlock
              ___
        ___
        chSysLock
```

7.18.4.16 chThdSleep()

```c
void chThdSleep (
    sysinterval_t time )
```

Suspends the invoking thread for the specified time.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>time</th>
<th>the delay in system ticks, the special values are handled as follow:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>• TIME_INFINITE the thread enters an infinite sleep state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• TIME_IMMEDIATE this value is not allowed.</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:

```
7.18.4.17 chThdSleepUntil()
void chThdSleepUntil (systime_t time)
Suspends the invoking thread until the system time arrives to the specified value.

Note
The function has no concept of "past", all specifiable times are in the future, this means that if you call this function exceeding your calculated intervals then the function will return in a far future time, not immediately.

See also
chThdSleepUntilWindowed()
```

Parameters

| in | time | absolute system time |
7.18 Threads

Function Class:

Normal API, 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)

### 7.18.4.18 chThdSleepUntilWindowed()

```
systime_t chThdSleepUntilWindowed (  
    systime_t prev,  
    systime_t next )
```

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

**Note**

The system time is assumed to be between `prev` and `next` else the call is assumed to have been called outside the allowed time interval, in this case no sleep is performed.

**See also**

- `chThdSleepUntil()`

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>prev</th>
<th>absolute system time of the previous deadline</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>next</td>
<td>absolute system time of the next deadline</td>
</tr>
</tbody>
</table>
Returns

the next parameter

Function Class:

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

Here is the call graph for this function:

7.18.4.19 chThdYield()

void chThdYield (void )

Yields the time slot.

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

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

Here is the call graph for this function:

7.18.4.20 chThdSuspendS()

msg_t chThdSuspendS (  
   thread_reference_t * trp )

Sends the current thread sleeping and sets a reference variable.

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

Parameters

| in | trp | a pointer to a thread reference object |

Returns
The wake up message.

Function Class:
This is an S-Class API, this function can be invoked from within a system lock zone by threads only.
Here is the call graph for this function:

```
  chThdSuspendS
  chSchGoSleepS
  chThdGetSelfX
  ch_pqueue_remove_highest
  chDbgCheckClassS
```

### 7.18.4.21 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>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>trp</td>
<td>a pointer to a thread reference object</td>
</tr>
<tr>
<td>in</td>
<td>timeout</td>
<td>the timeout in system ticks, the special values are handled as follow:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• TIME_INFINITE the thread enters an infinite sleep state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• TIME_IMMEDIATE the thread is not enqueued and the function returns</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MSG_TIMEOUT as if a timeout occurred.</td>
</tr>
</tbody>
</table>

**Returns**

The wake up message.

**Return values**

- **MSG_TIMEOUT** if the operation timed out.

**Function Class:**

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

### 7.18.4.22 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>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>in</code></td>
<td><code>trp</code> a pointer to a thread reference object</td>
</tr>
<tr>
<td><code>in</code></td>
<td><code>msg</code> 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:
7.18.4.23  chThdResumeS()

void chThdResumeS (  
    thread_reference_t * trp,  
    msg_t msg )

Wakes up a thread waiting on a thread reference object.

Note

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

Parameters

<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.

Here is the call graph for this function:

![Call Graph]

7.18.4.24  chThdResume()

void chThdResume (  
    thread_reference_t * trp,  
    msg_t msg )

Wakes up a thread waiting on a thread reference object.

Note

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

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>trp</td>
<td>a pointer to a thread reference object</td>
</tr>
<tr>
<td>in</td>
<td>msg</td>
<td>the message code</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](chart.png)

### 7.18.4.25 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

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>tqp</td>
<td>pointer to the threads queue object</td>
</tr>
<tr>
<td>in</td>
<td>timeout</td>
<td>the timeout in system ticks, the special values are handled as follow:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• TIME_INFINITE the thread enters an infinite sleep state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• TIME_IMMEDIATE the thread is not enqueued and the function returns MSG_TIMEOUT as if a timeout occurred.</td>
</tr>
</tbody>
</table>

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](image)

### 7.18.4.26 chThdDequeueNextI()

void chThdDequeueNextI (  
  threads_queue_t  *tqp,  
  msg_t msg  )

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

Parameters

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>tqp</td>
<td>pointer to the threads queue object</td>
</tr>
<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_graph_image)

### 7.18.4.27 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>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_graph_image)
7.18.4.28  chThdGetSelfX()

static thread_t* chThdGetSelfX (  
    void ) [inline], [static]

Returns a pointer to the current thread_t.

Returns

A pointer to the current thread.

Function Class:

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

7.18.4.29  chThdGetPriorityX()

static tprio_t chThdGetPriorityX (  
    void ) [inline], [static]

Returns the current thread priority.

Note

Can be invoked in any context.

Returns

The current thread priority.

Function Class:

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

Here is the call graph for this function:

7.18.4.30  chThdGetTicksX()

static systime_t chThdGetTicksX (  
    thread_t * tp ) [inline], [static]

Returns the number of ticks consumed by the specified thread.

Note

This function is only available when the CH_DBG_THREADS_PROFILING configuration option is enabled.
Parameters

\[
\text{in } \quad \text{tp} \quad \text{pointer to the thread}
\]

Returns

The number of consumed system ticks.

Function Class:

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

\subsection{7.18.4.31 \texttt{chThdGetWorkingAreaX()}}

```
static stkalign_t* chThdGetWorkingAreaX ( thread_t * tp ) [inline], [static]
```

Returns the working area base of the specified thread.

Parameters

\[
\text{in } \quad \text{tp} \quad \text{pointer to the thread}
\]

Returns

The working area base pointer.

Function Class:

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

\subsection{7.18.4.32 \texttt{chThdTerminatedX()}}

```
static bool chThdTerminatedX ( thread_t * tp ) [inline], [static]
```

Verifies if the specified thread is in the \texttt{CH\_STATE\_FINAL} state.

Parameters

\[
\text{in } \quad \text{tp} \quad \text{pointer to the thread}
\]
Return values

<table>
<thead>
<tr>
<th>true</th>
<th>thread terminated.</th>
</tr>
</thead>
<tbody>
<tr>
<td>false</td>
<td>thread not terminated.</td>
</tr>
</tbody>
</table>

Function Class:

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

### 7.18.4.33 chThdShouldTerminateX()

```c
static bool chThdShouldTerminateX ( void ) [inline], [static]
```

Verifies if the current thread has a termination request pending.

Return values

<table>
<thead>
<tr>
<th>true</th>
<th>termination request pending.</th>
</tr>
</thead>
<tbody>
<tr>
<td>false</td>
<td>termination request not pending.</td>
</tr>
</tbody>
</table>

Function Class:

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

Here is the call graph for this function:

```
chThdShouldTerminateX  
---------------------  
|                      |
|                      |
|                      |
|                      |
|                      |
|                      |
|                      |
|                      |
|                      |
|                      |
|                      |
|                      |
|                      |
|                      |
|                      |
|                      |
```

### 7.18.4.34 chThdStartI()

```c
static thread_t* chThdStartI ( thread_t * tp ) [inline], [static]
```

Resumes a thread created with `chThdCreateI()`.
7.18.4.35  chThdSleepS()

static void chThdSleepS (sysinterval_t ticks)  [inline], [static]

Suspends the invoking thread for the specified number of ticks.

Parameters

| in  | ticks | the delay in system ticks, the special values are handled as follow:
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>• TIME_INFINITE the thread enters an infinite sleep state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• TIME_IMMEDIATE this value is not allowed.</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)

### 7.18.4.36 chThdQueueObjectInit()

```c
static void chThdQueueObjectInit (threads_queue_t *tqp) [inline], [static]
```

Initializes a threads queue object.

**Parameters**

- `out 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.

Here is the call graph for this function:

![Call Graph](call_graph.png)

### 7.18.4.37 chThdQueueIsEmptyI()

```c
static bool chThdQueueIsEmptyI (threads_queue_t *tqp) [inline], [static]
```

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

```
out tqp pointer to the threads queue object
```

Returns

The queue status.

Return values

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

Here is the call graph for this function:

```
chThdQueueIsEmptyI
ch_queue_isempty
chDbgCheckClassI chSysHalt
```

7.18.4.38 chThdDoDequeueNextI()

```
static void chThdDoDequeueNextI (  
    threads_queue_t * tqp,   
    msg_t msg ) [inline], [static]
```

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

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

Precondition

The queue must contain at least an object.
Parameters

<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:
7.19 Time Measurement

7.19.1 Detailed Description

Time Measurement APIs and services.

Macros

- #define TM_CALIBRATION_LOOP 4U
  
  Number of iterations in the calibration loop.

Data Structures

- struct tm_calibration_t
  
  Type of a time measurement calibration data.

- struct time_measurement_t
  
  Type of a Time Measurement object.

Functions

- void _tm_init (void)
  
  Initializes the time measurement unit.

- void chTMObjectInit (time_measurement_t *tmp)
  
  Initializes a Time Measurement object.

- NOINLINE void chTMStartMeasurementX (time_measurement_t *tmp)
  
  Starts a measurement.

- NOINLINE void chTMStopMeasurementX (time_measurement_t *tmp)
  
  Stops a measurement.

- NOINLINE void chTMChainMeasurementToX (time_measurement_t *tmp1, time_measurement_t *tmp2)
  
  Stops a measurement and chains to the next one using the same time stamp.

7.19.2 Macro Definition Documentation

7.19.2.1 TM_CALIBRATION_LOOP

#define TM_CALIBRATION_LOOP 4U

Number of iterations in the calibration loop.

Note

This is required in order to assess the best result in architectures with instruction cache.
7.19.3 Function Documentation

7.19.3.1 _tm_init()

```c
void _tm_init (  
    void  )
```

Initializes the time measurement unit.

Function Class:

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

Here is the call graph for this function:

```
  _tm_init      
      |        
  +----------------------+
  |                      |
  | chTMObjectInit       |
  |                      |
  +----------------------+
    |            |
    |            |
    | chTMStart MeasurementX |
    |                      |
    |                      |
    +----------------------+
      |        
    chTMStopMeasurementX
```

7.19.3.2 chTMObjectInit()

```c
void chTMObjectInit (  
    time_measurement_t * tmp  )
```

Initializes a TimeMeasurement object.

Parameters

| out | tmp | pointer to a TimeMeasurement structure |

Function Class:

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.
7.19.3.3 **chTMStartMeasurementX()**

```c
NOINLINE void chTMStartMeasurementX ( 
    time_measurement_t * tmp )
```

Starts a measurement.

**Precondition**

The `time_measurement_t` structure must be initialized.

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>in,out</code></td>
<td><code>tmp</code></td>
<td>pointer to a <code>TimeMeasurement</code> structure</td>
</tr>
</tbody>
</table>

**Function Class:**

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

7.19.3.4 **chTMStopMeasurementX()**

```c
NOINLINE void chTMStopMeasurementX ( 
    time_measurement_t * tmp )
```

Stops a measurement.

**Precondition**

The `time_measurement_t` structure must be initialized.

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>in,out</code></td>
<td><code>tmp</code></td>
<td>pointer to a <code>time_measurement_t</code> structure</td>
</tr>
</tbody>
</table>

**Function Class:**

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

7.19.3.5 **chTMChainMeasurementToX()**

```c
NOINLINE void chTMChainMeasurementToX ( 
    time_measurement_t * tmp1, 
    time_measurement_t * tmp2 )
```

Stops a measurement and chains to the next one using the same time stamp.
Parameters

<table>
<thead>
<tr>
<th>in, out</th>
<th>tmp1</th>
<th>pointer to the <code>time_measurement_t</code> structure to be stopped</th>
</tr>
</thead>
<tbody>
<tr>
<td>in, out</td>
<td>tmp2</td>
<td>pointer to the <code>time_measurement_t</code> structure to be started</td>
</tr>
</tbody>
</table>

Function Class:

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

7.20.1 Detailed Description

Synchronization services.

**Modules**

- Counting Semaphores
- Mutexes
- Condition Variables
- Event Flags
- Synchronous Messages
7.21 Counting Semaphores

7.21.1 Detailed Description

Semaphores related APIs and services.

Operation mode

Semaphores are a flexible synchronization primitive, ChibiOS/RT implements semaphores in their "counting semaphores" variant as defined by Edsger Dijkstra plus several enhancements like:

- Wait operation with timeout.
- Reset operation.
- Atomic wait+signal operation.
- Return message from the wait operation (OK, RESET, TIMEOUT).

The binary semaphores variant can be easily implemented using counting semaphores.

Operations defined for semaphores:

- **Signal**: The semaphore counter is increased and if the result is non-positive then a waiting thread is removed from the semaphore queue and made ready for execution.
- **Wait**: The semaphore counter is decreased and if the result becomes negative the thread is queued in the semaphore and suspended.
- **Reset**: The semaphore counter is reset to a non-negative value and all the threads in the queue are released.

Semaphores can be used as guards for mutual exclusion zones (note that mutexes are recommended for this kind of use) but also have other uses, queues guards and counters for example. Semaphores usually use a FIFO queuing strategy but it is possible to make them order threads by priority by enabling `CH_CFG_USE_SEMAPHORES_PRIORITY` in `chconf.h`.

Precondition

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

Macros

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

Typedefs

- `typedef struct ch_semaphore semaphore_t`
  Semaphore structure.
Data Structures

- struct ch_semaphore
  
  Semaphore structure.

Functions

- void chSemObjectInit (semaphore_t *sp, cnt_t n)
  
  Initializes a semaphore with the specified counter value.

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

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

- msg_t chSemWait (semaphore_t *sp)
  
  Performs a wait operation on a semaphore.

- msg_t chSemWaitS (semaphore_t *sp)
  
  Performs a wait operation on a semaphore.

- msg_t chSemWaitTimeout (semaphore_t *sp, sysinterval_t timeout)
  
  Performs a wait operation on a semaphore with timeout specification.

- msg_t chSemWaitTimeoutS (semaphore_t *sp, sysinterval_t timeout)
  
  Performs a wait operation on a semaphore with timeout specification.

- void chSemSignal (semaphore_t *sp)
  
  Performs a signal operation on a semaphore.

- void chSemSignalI (semaphore_t *sp)
  
  Performs a signal operation on a semaphore.

- void chSemAddCounterI (semaphore_t *sp, cnt_t n)
  
  Adds the specified value to the semaphore counter.

- msg_t chSemSignalWait (semaphore_t *sp, semaphore_t *spw)
  
  Performs atomic signal and wait operations on two semaphores.

- static void chSemReset (semaphore_t *sp, cnt_t n)
  
  Performs a reset operation on the semaphore.

- static void chSemResetI (semaphore_t *sp, cnt_t n)
  
  Performs a reset operation on the semaphore.

- static void chSemFastWaitI (semaphore_t *sp)
  
  Decreases the semaphore counter.

- static void chSemFastSignalI (semaphore_t *sp)
  
  Increases the semaphore counter.

- static cnt_t chSemGetCounterI (const semaphore_t *sp)
  
  Returns the semaphore counter current value.

7.21.2 Macro Definition Documentation

7.21.2.1 _SEMAPHORE_DATA

#define _SEMAPHORE_DATA(
  name,
  n) ({_CH_QUEUE_DATA(name.queue), n})

Data part of a static semaphore initializer.

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

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

### 7.21.2.2 SEMAPHORE_DECL

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

### 7.21.3 Typedef Documentation

#### 7.21.3.1 semaphore_t

```c
typedef struct ch_semaphore semaphore_t
```

Semaphore structure.

### 7.21.4 Function Documentation

#### 7.21.4.1 chSemObjectInit()

```c
void chSemObjectInit ( 
    semaphore_t * sp, 
    cnt_t 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>

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:

```
chSemObjectInit → ch_queue_init
```

### 7.21.4.2 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.

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>

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

Here is the call graph for this function:

7.21.4.3 chSemResetWithMessageI()

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

Performs a reset operation on the semaphore.

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

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

Parameters

| in | sp | pointer to a semaphore_t structure |
| in | n | the new value of the semaphore counter. The value must be non-negative. |
| in | msg | message to be sent |
Function Class:

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

Here is the call graph for this function:

![Call Graph](chSemWait.png)

### 7.21.4.4 chSemWait()

```c
msg_t chSemWait (semaphore_t * sp )
```

Performs a wait operation on a semaphore.

**Parameters**

- `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>MSG_OK</th>
<th>if the thread has not stopped on the semaphore or the semaphore has been signaled.</th>
</tr>
</thead>
</table>
| MSG_RESET | if the semaphore has been reset using `chSemReset()`.
|
Function Class:
Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

![Call Graph Image]

### 7.21.4.5 chSemWaitS()

```c
msg_t chSemWaitS ( 
    semaphore_t * sp )
```

**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>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>MSG_RESET</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.

Here is the call graph for this function:

```
ch_queue isempty
chSemWaitS
ch_queue notempty
chDbgCheckClassS
chSysHalt
```

### 7.21.4.6 chSemWaitTimeout()

```c
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 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

| MSG_OK          | if the thread has not stopped on the semaphore or the semaphore has been signaled. |
| MSG_RESET       | if the semaphore has been reset using `chSemReset()`.                           |
| MSG_TIMEOUT     | if the semaphore has not been signaled or reset within the specified timeout.   |

Function Class:

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

Here is the call graph for this function:

7.21.4.7 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

| MSG_OK          | if the thread has not stopped on the semaphore or the semaphore has been signaled. |
| MSG_RESET       | if the semaphore has been reset using chSemReset(). |
| MSG_TIMEOUT     | if the semaphore has not been signaled or reset within the specified timeout. |

Function Class:

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

Here is the call graph for this function:

```
ch_queue_isempty

chSemWaitTimeoutS

ch_queue_notempty

chDbgCheckClassS

chSysHalt
```

7.21.4.8 chSemSignal()

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

Performs a signal operation on a semaphore.

Parameters

- **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:

```
7.21.4.9 chSemSignalI()

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
```
7.21 Counting Semaphores

Function Class:

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

Here is the call graph for this function:

7.21.4.10 chSemAddCounterI()

```c
void chSemAddCounterI (
    semaphore_t * sp,
    cnt_t n )
```

Adds the specified value to the semaphore counter.

Postcondition

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

Parameters

<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>value to be added to the semaphore counter. The value must be positive.</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:

```
chSemAddCounterI
   |   \ ch_queue_fifo_remove
   |    \ ch_queue_isempty
   |     \ ch_queue_notempty
   |     \ chDbgCheckClassI
   |     \ chSchReadyI
   |     \ ch_pqueue_insert_behind
    \ chSysHalt
```

7.21.4.11  chSemSignalWait()

```c
msg_t chSemSignalWait ( 
    semaphore_t * sps, 
    semaphore_t * spw )
```

Performs atomic signal and wait operations on two semaphores.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>sps</th>
<th>pointer to a semaphore_t structure to be signaled</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>spw</td>
<td>pointer to a semaphore_t structure to be wait on</td>
</tr>
</tbody>
</table>

**Returns**

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

**Return values**

| MSG_OK          | if the thread has not stopped on the semaphore or the semaphore has been signaled. |
| MSG_RESET       | if the semaphore has been reset using chSemReset (). |
Function Class:

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

Here is the call graph for this function:

```
chSemSignalWait
  ch_queue_lifo_remove
    ch_queue_sempy
  ch_queue_notempty
    ch_pqueue_insert_behind
      chDbgCheckClass
        _dbg_check_lock
          _stats_start_measure
            _cnt_thd

chSchReady
  chQueueRemove
    chQueueIsEmpty
      chQueueNotEmpty
```

### 7.21.4.12 chSemReset()

```c
static void chSemReset (
  semaphore_t * sp,
  cnt_t n ) [inline], [static]
```

Performs a reset operation on the semaphore.

**Postcondition**

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

**Note**

This function implicitly sends MSG_RESET as message.

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>sp</td>
</tr>
<tr>
<td>in</td>
<td>n</td>
</tr>
</tbody>
</table>
212 Module Documentation

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

Here is the call graph for this function:

```
chSemReset
chSemResetWithMessage
chSchRescheduleS
chSemResetWithMessageI
chSysLock
chSysUnlock
```

7.21.4.13  chSemResetI()

```c
static void chSemResetI (semaphore_t * sp, cnt_t n) [inline], [static]
```

Performs a reset operation on the semaphore.

Postcondition

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

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

Note

This function implicitly sends MSG_RESET as message.

Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>in</code></td>
<td><code>sp</code></td>
</tr>
<tr>
<td><code>in</code></td>
<td><code>n</code></td>
</tr>
</tbody>
</table>
7.21 Counting Semaphores

Function Class:

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

Here is the call graph for this function:

7.21.4.14 chSemFastWaitI()

static void chSemFastWaitI ( 
   semaphore_t * sp ) [inline], [static]

Decreases the semaphore counter.

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

Parameters

| in | sp | pointer to a semaphore_t structure |

Function Class:

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

Here is the call graph for this function:
7.21.4.15  chSemFastSignalI()

static void chSemFastSignalI ( 
    semaphore_t * sp ) [inline], [static]

 Increases the semaphore counter.

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

 Parameters

| in  | sp | pointer to a semaphore_t structure |

 Function Class:

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

 Here is the call graph for this function:

7.21.4.16  chSemGetCounterI()

static cnt_t chSemGetCounterI ( 
    const semaphore_t * sp ) [inline], [static]

 Returns the semaphore counter current value.

 Parameters

| in  | sp | pointer to a semaphore_t structure |

 Returns

 The semaphore counter value.
Function Class:

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

Here is the call graph for this function:
7.22 Mutexes

7.22.1 Detailed Description

Mutexes related APIs and services.

Operation mode

A mutex is a threads synchronization object that can be in two distinct states:

- Not owned (unlocked).
- Owned by a thread (locked).

Operations defined for mutexes:

- **Lock**: The mutex is checked, if the mutex is not owned by some other thread then it is associated to the
  locking thread else the thread is queued on the mutex in a list ordered by priority.
- **Unlock**: The mutex is released by the owner and the highest priority thread waiting in the queue, if any, is
  resumed and made owner of the mutex.

Constraints

In ChibiOS/RT the Unlock operations must always be performed in lock-reverse order. This restriction both improves
the performance and is required for an efficient implementation of the priority inheritance mechanism. Operating under
this restriction also ensures that deadlocks are no possible.

Recursive mode

By default mutexes are not recursive, this mean that it is not possible to take a mutex already owned by the same
thread. It is possible to enable the recursive behavior by enabling the option `CH_CFG_USE_MUTEXES_RECURSIVE`.

The priority inversion problem

The mutexes in ChibiOS/RT implements the **full** priority inheritance mechanism in order handle the priority inversion
problem. When a thread is queued on a mutex, any thread, directly or indirectly, holding the mutex gains the same priority
of the waiting thread (if their priority was not already equal or higher). The mechanism works with any number of
nested mutexes and any number of involved threads. The algorithm complexity (worst case) is $N$ with $N$ equal to
the number of nested mutexes.

Precondition

In order to use the mutex APIs the `CH_CFG_USE_MUTEXES` option must be enabled in `chconf.h`.

Postcondition

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

Macros

- `#define _MUTEX_DATA(name) (_CH_QUEUE_DATA(name.queue), NULL, NULL, 0)`
  Data part of a static mutex initializer.
- `#define MUTEX_DECL(name) mutex_t name = _MUTEX_DATA(name)`
  Static mutex initializer.

Typedefs

- `typedef struct ch_mutex mutex_t`
  Type of a mutex structure.

Data Structures

- `struct ch_mutex`
  Mutex structure.

Functions

- `void chMtxObjectInit (mutex_t *mp)`
  Initializes a mutex_t structure.
- `void chMtxLock (mutex_t *mp)`
  Locks the specified mutex.
- `void chMtxLockS (mutex_t *mp)`
  Locks the specified mutex.
- `bool chMtxTryLock (mutex_t *mp)`
  Tries to lock a mutex.
- `bool chMtxTryLockS (mutex_t *mp)`
  Tries to lock a mutex.
- `void chMtxUnlock (mutex_t *mp)`
  Unlocks the specified mutex.
- `void chMtxUnlockS (mutex_t *mp)`
  Unlocks the specified mutex.
- `void chMtxUnlockAllS (void)`
  Unlocks all mutexes owned by the invoking thread.
- `void chMtxUnlockAll (void)`
  Unlocks all mutexes owned by the invoking thread.
- `static bool chMtxQueueNotEmptyS (mutex_t *mp)`
  Returns true if the mutex queue contains at least a waiting thread.
- `static thread_t * chMtxGetOwnerI (mutex_t *mp)`
  Returns the mutex owner thread.
- `static mutex_t * chMtxGetNextMutexX (void)`
  Returns the next mutex in the mutexes stack of the current thread.

7.22.2 Macro Definition Documentation
7.22.2.1 _MUTEX_DATA

#define _MUTEX_DATA(
    name ) {CH_QUEUE_DATA(name.queue), NULL, NULL, 0}

Data part of a static mutex initializer.

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

| in  | name | the name of the mutex variable |

### 7.22.2 MUTEX_DECL

```c
#define MUTEX_DECL(name) mutex_t name = _MUTEX_DATA(name)
```

Static mutex initializer.

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

**Parameters**

| in  | name | the name of the mutex variable |

### 7.22.3 Typedef Documentation

#### 7.22.3.1 mutex_t

```c
typedef struct ch_mutex mutex_t
```

Type of a mutex structure.

### 7.22.4 Function Documentation

#### 7.22.4.1 chMtxObjectInit()

```c
void chMtxObjectInit ( mutex_t *mp )
```

Initializes a `mutex_t` structure.

**Parameters**

| out  | mp   | pointer to a `mutex_t` structure |
Function Class:
 Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

Here is the call graph for this function:

```
chMtxObjectInit → ch_queue_init
```

7.22.4.2 chMtxLock()

```c
void chMtxLock ( mutex_t * mp )

Locks the specified mutex.

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

Parameters

| in  | mp   | pointer to the mutex_t structure |

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

Here is the call graph for this function:

![Call Graph Image]

### 7.22.4.3 chMtxLockS()

```c
void chMtxLockS(
    mutex_t * mp
)
```

Locks the specified mutex.

**Postcondition**

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

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>mp</th>
<th>pointer to the mutex_t structure</th>
</tr>
</thead>
</table>
7.22.4.4 chMtxTryLock()

bool chMtxTryLock (  
    mutex_t * mp  )

Tries to lock a mutex.

This function attempts to lock a mutex, if the mutex is already locked by another thread then the function exits without waiting.

Postcondition

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

Note

This function does not have any overhead related to the priority inheritance mechanism because it does not try to enter a sleep state.

Parameters

| in | mp | pointer to the mutex_t structure |

Returns

The operation status.
Return values

<table>
<thead>
<tr>
<th>true</th>
<th>if the mutex has been successfully acquired</th>
</tr>
</thead>
<tbody>
<tr>
<td>false</td>
<td>if the lock attempt 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.

Here is the call graph for this function:

```
chMtxTryLock
    o chMtxTryLockS
        o chDbgCheckClassS
        o chThdGetSelfX
    o _dbg_check_lock
    o _stats_start_measure
    o _crit_thd
    o _stats_stop_measure
    o _crit_thd
    o chSysUnlock
        o _dbg_check_unlock
        o _stats_start_measure
        o _crit_thd
    o chSysLock
        o _stats_start_measure
        o _crit_thd
```

### 7.22.4.5  chMtxTryLockS()

```c
bool chMtxTryLockS ( mutex_t * mp )
```

Tries to lock a mutex.

This function attempts to lock a mutex, if the mutex is already taken by another thread then the function exits without waiting.

Postcondition

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

Note

This function does not have any overhead related to the priority inheritance mechanism because it does not try to enter a sleep state.
Parameters

| in | mp | pointer to the mutex_t structure |

Returns

The operation status.

Return values

| true | if the mutex has been successfully acquired |
| false | if the lock attempt failed. |

Function Class:

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

Here is the call graph for this function:

```
chMtxTryLockS
  `-- chDbgCheckClassS
                                `-- chSysHalt
             chMtxTryLockS
                     `-- chThdGetSelfX
```

7.22.4.6 chMtxUnlock()

```c
void chMtxUnlock ( 
                mutex_t * mp )
```

Unlocks the specified mutex.

Note

Mutexes must be unlocked in reverse lock order. Violating this rules will result in a panic if assertions are enabled.

Precondition

The invoking thread **must** have at least one owned mutex.

Postcondition

The mutex is unlocked and removed from the per-thread stack of owned mutexes.
7.22 Mutexes

Parameters

\[ \text{in} \ mp \ \text{pointer to the} \ mutex\_t \text{ structure} \]

Function Class:

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

Here is the call graph for this function:

```
7.22.4.7 chMtxUnlockS()

void chMtxUnlockS ( mutex_t * mp )

Unlocks the specified mutex.

Note

Mutexes must be unlocked in reverse lock order. Violating this rules will result in a panic if assertions are enabled.

Precondition

The invoking thread must have at least one owned mutex.

Postcondition

The mutex is unlocked and removed from the per-thread stack of owned mutexes. This function does not reschedule so a call to a rescheduling function must be performed before unlocking the kernel.

ChibiOS/RT
Parameters

in  *mp*  pointer to the `mutex_t` structure

Function Class:

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

Here is the call graph for this function:

![Call Graph]

### 7.22.4.8  chMtxUnlockAllS()

```c
void chMtxUnlockAllS (  
    void  )
```

Unlocks all mutexes owned by the invoking thread.

**Postcondition**

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

This function does not reschedule so a call to a rescheduling function must be performed before unlocking the kernel.

**Note**

This function is **MUCH MORE** efficient than releasing the mutexes one by one and not just because the call overhead, this function does not have any overhead related to the priority inheritance mechanism.
Function Class:

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

Here is the call graph for this function:

![Call Graph Image](https://via.placeholder.com/150)

### 7.22.4.9 chMtxUnlockAll()

```c
void chMtxUnlockAll ( void )
```

Unlocks all mutexes owned by the invoking thread.

**Postcondition**

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

**Note**

This function is **MUCH MORE** efficient than releasing the mutexes one by one and not just because the call overhead, this function does not have any overhead related to the priority inheritance mechanism.
Function Class:
Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

![Call Graph Diagram]

### 7.22.4.10 chMtxQueueNotEmptyS()

```c
static bool chMtxQueueNotEmptyS ( mutex_t * mp ) [inline], [static]
```

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

**Parameters**
- **out mp** pointer to a mutex_t structure

**Returns**
The mutex queue status.
Function Class:
This is an **S-Class** API, this function can be invoked from within a system lock zone by threads only.

Here is the call graph for this function:

```
chMtxQueueNotEmptyS
  |            |
  v            |
ch_queue_notempty
  |            |
  v            |
chDbgCheckClassS
  |            |
  v            |
  |            |
  chSysHalt
```

### 7.22.4.11 chMtxGetOwnerI()

```c
static thread_t* chMtxGetOwnerI ( mutex_t * mp ) [inline], [static]
```

Returns the mutex owner thread.

**Parameters**

| out | mp   | pointer to a mutex_t structure |

**Return values**

- **NULL** if the mutex is not owned.
Function Class:

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

Here is the call graph for this function:

![Call Graph]

### 7.22.4.12 chMtxGetNextMutexX()

```c
static mutex_t* chMtxGetNextMutexX ( 
    void ) [inline], [static]
```

Returns the next mutex in the mutexes stack of the current thread.

Returns

A pointer to the next mutex in the stack.

Return values

| NULL   | if the stack is empty. |

Function Class:

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

Here is the call graph for this function:

![Call Graph]
7.23 Condition Variables

7.23.1 Detailed Description

This module implements the Condition Variables mechanism. Condition variables are an extensions to the mutex subsystem and cannot work alone.

Operation mode

The condition variable is a synchronization object meant to be used inside a zone protected by a mutex. Mutexes and condition variables together can implement a Monitor construct.

Precondition

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

Macros

- \#define _CONDVAR_DATA(name) (_CH_QUEUE_DATA(name.queue))
  
  Data part of a static condition variable initializer.

- \#define CONDVAR_DECL(name) condition_variable_t name = _CONDVAR_DATA(name)
  
  Static condition variable initializer.

Typedefs

- typedef struct condition_variable condition_variable_t

  condition_variable_t structure.

Data Structures

- struct condition_variable

  condition_variable_t structure.

Functions

- void chCondObjectInit (condition_variable_t *cp)
  
  Initializes a condition_variable_t structure.

- void chCondSignal (condition_variable_t *cp)
  
  Signals one thread that is waiting on the condition variable.

- void chCondSignalI (condition_variable_t *cp)
  
  Signals one thread that is waiting on the condition variable.

- void chCondBroadcast (condition_variable_t *cp)
  
  Signals all threads that are waiting on the condition variable.

- void chCondBroadcastI (condition_variable_t *cp)
  
  Signals all threads that are waiting on the condition variable.

- msg_t chCondWait (condition_variable_t *cp)
  
  Waits on the condition variable releasing the mutex lock.

- msg_t chCondWaitS (condition_variable_t *cp)
  
  Waits on the condition variable releasing the mutex lock.

- msg_t chCondWaitTimeout (condition_variable_t *cp, sysinterval_t timeout)
  
  Waits on the condition variable releasing the mutex lock.

- msg_t chCondWaitTimeoutS (condition_variable_t *cp, sysinterval_t timeout)
  
  Waits on the condition variable releasing the mutex lock.
7.23.2 Macro Definition Documentation

7.23.2.1 _CONDVAR_DATA

#define _CONDVAR_DATA(name) {_CH_QUEUE_DATA(name.queue)}

Data part of a static condition variable initializer.

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

Parameters

| in | name | the name of the condition variable |

7.23.2.2 CONDVAR_DECL

#define CONDVAR_DECL(name) condition_variable_t name = _CONDVAR_DATA(name)

Static condition variable initializer.

Statically initialized condition variables require no explicit initialization using chCondInit().

Parameters

| in | name | the name of the condition variable |

7.23.3 Typedef Documentation

7.23.3.1 condition_variable_t

typedef struct condition_variable condition_variable_t

condition_variable_t structure.

7.23.4 Function Documentation
7.23.4.1 chCondObjectInit()

```c
void chCondObjectInit ( 
    condition_variable_t * cp )
```

Initializes a `condition_variable_t` structure.

**Parameters**

- `out cp` pointer to a `condition_variable_t` structure

**Function Class:**

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

Here is the call graph for this function:

![Call Graph]

7.23.4.2 chCondSignal()

```c
void chCondSignal ( 
    condition_variable_t * cp )
```

Signals one thread that is waiting on the condition variable.

**Parameters**

- `in cp` pointer to the `condition_variable_t` structure
Function Class:

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

Here is the call graph for this function:

7.23.4.3 chCondSignalI()

void chCondSignalI (condition_variable_t * cp )

Signals one thread that is waiting on the condition variable.

Postcondition

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

Parameters

| in | cp | pointer to the condition_variable_t structure |
Function Class:

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

Here is the call graph for this function:

![Call Graph](image)

### 7.23.4.4 chCondBroadcast()

```c
void chCondBroadcast ( condition_variable_t * cp )
```

Signals all threads that are waiting on the condition variable.

**Parameters**

| in | cp | pointer to the condition_variable_t structure |

Function Class:

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

```
7.23.4.5 chCondBroadcastI()

void chCondBroadcastI (  
    condition_variable_t * cp  )  
```

Signals all threads that are waiting on the condition variable.

**Postcondition**

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

**Parameters**

| in | cp | pointer to the condition_variable_t structure |

**Function Class:**

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

![Call Graph]

### 7.23.4.6 chCondWait()

```c
cmsg_t chCondWait (condition_variable_t * cp)
```

Waits on the condition variable releasing the mutex lock.

Releases the currently owned mutex, waits on the condition variable, and finally acquires the mutex again. All the sequence is performed atomically.

**Precondition**

The invoking thread **must** have at least one owned mutex.

**Parameters**

- **in cp** pointer to the `condition_variable_t` structure

**Returns**

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

**Return values**

- **MSG_OK** if the condition variable has been signaled using `chCondSignal()`.
- **MSG_RESET** if the condition variable has been signaled using `chCondBroadcast()`.
7.23.4.7 chCondWaitS()

msg_t chCondWaitS (condition_variable_t * cp )

Waits on the condition variable releasing the mutex lock.

Releases the currently owned mutex, waits on the condition variable, and finally acquires the mutex again. All the sequence is performed atomically.

Precondition

The invoking thread **must** have at least one owned mutex.

Parameters

| in | cp | pointer to the condition_variable_t structure |

Returns

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

<table>
<thead>
<tr>
<th>MSG_OK</th>
<th>if the condition variable has been signaled using chCondSignal().</th>
</tr>
</thead>
</table>
| MSG_RESET | if the condition variable has been signaled using chCondBroadca

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)

### 7.23.4.8 chCondWaitTimeout()

```c
msg_t chCondWaitTimeout (  
    condition_variable_t * cp,  
    sysinterval_t timeout )
```

Waits on the condition variable releasing the mutex lock.

Releases the currently owned mutex, waits on the condition variable, and finally acquires the mutex again. All the sequence is performed atomically.

Precondition

The invoking thread **must** have at least one owned mutex.

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

Postcondition

Exiting the function because a timeout does not re-acquire the mutex, the mutex ownership is lost.
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>cp</code></td>
<td>pointer to the <code>condition_variable_t</code> structure</td>
</tr>
<tr>
<td><code>timeout</code></td>
<td>the number of ticks before the operation timeouts, the special values are handled as follow:</td>
</tr>
<tr>
<td></td>
<td>• <code>TIME_INFINITE</code> no timeout.</td>
</tr>
<tr>
<td></td>
<td>• <code>TIME_IMMEDIATE</code> this value is not allowed.</td>
</tr>
</tbody>
</table>

Returns

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

Return values

| MSG_OK | if the condition variable has been signaled using `chCondSignal()`. |
| MSG_RESET | if the condition variable has been signaled using `chCondBroadcast()`. |
| MSG_TIMEOUT | if the condition variable has not been signaled within the specified timeout. |

Function Class:

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

Here is the call graph for this function:
7.23 Condition Variables

7.23.4.9 chCondWaitTimeoutS()

```c
msg_t chCondWaitTimeoutS (
    condition_variable_t * cp,
    sysinterval_t timeout
)
```

Waits on the condition variable releasing the mutex lock.

Releases the currently owned mutex, waits on the condition variable, and finally acquires the mutex again. All the sequence is performed atomically.

Precondition

The invoking thread **must** have at least one owned mutex.

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

Postcondition

Exiting the function because a timeout does not re-acquire the mutex, the mutex ownership is lost.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>cp</th>
<th>pointer to the condition_variable_t structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>timeout</td>
<td>the number of ticks before the operation timeouts, the special values are handled as follow:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• TIME_INFINITE no timeout.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• TIME_IMMEDIATE this value is not allowed.</td>
</tr>
</tbody>
</table>

Returns

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

Return values

<table>
<thead>
<tr>
<th>MSG_OK</th>
<th>if the condition variable has been signaled using chCondSignal().</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSG_RESET</td>
<td>if the condition variable has been signaled using chCondBroadcast().</td>
</tr>
<tr>
<td>MSG_TIMEOUT</td>
<td>if the condition variable has not been signaled 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:
7.24 Event Flags

7.24.1 Detailed Description

Event Flags, Event Sources and Event Listeners.

Operation mode

Each thread has a mask of pending events inside its `thread_t` structure. Operations defined for events:

- **Wait**, the invoking thread goes to sleep until a certain AND/OR combination of events are signaled.
- **Clear**, a mask of events is cleared from the pending events, the cleared events mask is returned (only the events that were actually pending and then cleared).
- **Signal**, an events mask is directly ORed to the mask of the signaled thread.
- **Broadcast**, each thread registered on an Event Source is signaled with the events specified in its Event Listener.
- **Dispatch**, an events mask is scanned and for each bit set to one an associated handler function is invoked. Bit masks are scanned from bit zero upward.

An Event Source is a special object that can be "broadcasted" by a thread or an interrupt service routine. Broadcasting an Event Source has the effect that all the threads registered on the Event Source will be signaled with an events mask.

An unlimited number of Event Sources can exists in a system and each thread can be listening on an unlimited number of them.

Precondition

In order to use the Events APIs the `CH_CFG_USE_EVENTS` option must be enabled in `chconf.h`.

Postcondition

Enabling events requires 1-4 (depending on the architecture) extra bytes in the `thread_t` structure.

Macros

- `#define ALL_EVENTS ((eventmask_t)-1)`
  
  All events allowed mask.
- `#define EVENT_MASK(eid) ((eventmask_t)(1<<eventmask_t(eid)))`
  
  Returns an event mask from an event identifier.
- `#define _EVENTSOURCE_DATA(name) {(event_listener_t *)&name}`
  
  Data part of a static event source initializer.
- `#define EVENTSOURCE_DECL(name) event_source_t name = _EVENTSOURCE_DATA(name)`
  
  Static event source initializer.
Typedefs

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

Data Structures

- struct event_listener
  Event Listener structure.
- struct event_source
  Event Source structure.

Functions

- void 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 chEvtWaitOne (eventmask_t events)
  Waits for exactly one of the specified events.
- eventmask_t chEvtWaitAny (eventmask_t events)
  Waits for any of the specified events.
- eventmask_t chEvtWaitAll (eventmask_t events)
  Waits for all the specified events.
- eventmask_t chEvtWaitOneTimeout (eventmask_t events, sysinterval_t timeout)
  Waits for exactly one of the specified events.
7.24 Event Flags

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

- `eventmask_t chEvtWaitAllTimeout (eventmask_t events, sysinterval_t timeout)`

  Waits for all the specified events.

- `static void chEvtObjectInit (event_source_t *esp)`

  Initializes an Event Source.

- `static void chEvtRegisterMask (event_source_t *esp, event_listener_t *elp, eventmask_t events)`

  Registers an Event Listener on an Event Source.

- `static void chEvtRegister (event_source_t *esp, event_listener_t *elp, eventid_t event)`

  Registers an Event Listener on an Event Source.

- `static bool chEvtIsListeningI (event_source_t *esp)`

  Verifies if there is at least one `event_listener_t` registered.

- `static void chEvtBroadcast (event_source_t *esp)`

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

- `static void chEvtBroadcastI (event_source_t *esp)`

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

- `static eventmask_t chEvtAddEventsI (eventmask_t events)`

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

- `static eventmask_t chEvtGetEventsX (void)`

  Returns the events mask.

### 7.24.2 Macro Definition Documentation

#### 7.24.2.1 ALL_EVENTS

```c
#define ALL_EVENTS ((eventmask_t)-1)
```

All events allowed mask.

#### 7.24.2.2 EVENT_MASK

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

Returns an event mask from an event identifier.

#### 7.24.2.3 _EVENTSOURCE_DATA

```c
#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 |

7.24.2.4 EVENTSOURCE_DECL

```c
#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 |

7.24.3 Typedef Documentation

7.24.3.1 event_source_t

typedef struct event_source event_source_t

Event Source structure.

7.24.3.2 evhandler_t

typedef void(* evhandler_t) (eventid_t id)

Event Handler callback function.

7.24.4 Function Documentation
7.24.4.1 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.

Here is the call graph for this function:

7.24.4.2 chEvtUnregister()

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

Unregisters an Event Listener from its Event Source.
If the event listener is not registered on the specified event source then the function does nothing.

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

Parameters

| in | esp | pointer to the event_source_t structure |
| in | elp | pointer to the event_listener_t structure |

Function Class:

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

Here is the call graph for this function:

7.24.4.3 chEvtGetAndClearEventsI()

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.

### 7.24.4 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:
### 7.24.4.5 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**
- `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.

Here is the call graph for this function:

[Call graph diagram]

### 7.24.4.6 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.
7.24 Event Flags

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:

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:

```
chEvtBroadcastFlags
chEvtSignal
chEvtGetAndClearFlags
chSchReady
chDbgCheckClass
chSysHalt
```

7.24.4.7 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.

Here is the call graph for this function:

7.24.4.8 chEvtSignal()

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

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

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td><code>tp</code></td>
</tr>
<tr>
<td>in</td>
<td><code>events</code></td>
</tr>
</tbody>
</table>
7.24 Event Flags

Function Class:

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

Here is the call graph for this function:

```
chEvtSignal
  ↓
chEvtSignalI
  ↓
chSchRescheduleS
  ↓
chSysLock
  ↓
chSysUnlock
  ↓
chDbgCheckClassI
  ↓
chSchReadyI
  ↓
chDbgCheckClassS
  ↓
chSchDoRescheduleAhead
  ↓
chSchIsRescRequiredI
  ↓
_dbg_check_lock
  ↓
_stats_start_measure
  ↓._crit_thd
  ↓
_dbg_check_unlock
  ↓
_stats_stop_measure
  ↓._crit_thd
```

7.24.4.9 chEvtSignalI()

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

Adding 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

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in tp</td>
<td>the thread to be signaled</td>
</tr>
<tr>
<td>in events</td>
<td>the events set to be ORed</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)

### 7.24.4.10 chEvtBroadcastFlags()

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

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

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

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>esp</th>
<th>pointer to the <code>event_source_t</code> 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.
7.24 Event Flags

Here is the call graph for this function:

```
    chEvtBroadcastFlags
    chEvtBroadcastFlagsI
    chSchRescheduleS
    chSysLock
    chSysUnlock
    chDbgCheckClassI
    chEvtSignalI
    chDbgCheckClassS
    chSchDoRescheduleAhead
    _dbg_check_lock
    _stats_start_measure
    _crit_thd
    _dbg_check_unlock
    _stats_stop_measure
    _crit_thd
```

7.24.4.11 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.
7.24.4.12 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.

7.24.4.13 chEvtWaitOne()

eventmask_t chEvtWaitOne (  
    eventmask_t events )

Waits for exactly one of the specified events.

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

Note

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

Parameters

| in | 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.
Function Class:

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

Here is the call graph for this function:

```
chEvtWaitOne
  chSchGoSleepS
    chDbgCheckClassS
      _dbg_check_lock
        _stats_start_measure
          _crit_thd
      _dbg_check_unlock
        _stats_stop_measure
          _crit_thd
  chSysLock
    ch_pqueue_remove_highest

7.24.4.14  chEvtWaitAny()

eventmask_t chEvtWaitAny (  
    eventmask_t events  )

Waits for any of the specified events.

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

Parameters

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

Returns

The mask of the served and cleared events.

Function Class:

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

```
chEvtWaitAny
   |<------------------
   |                chDbgCheckClassS
   |                 
   |                 _dbg_check_unlock
   |                 
   |                 _stats_stop_measure
   |                 _crit_thd
   |                 
   | ch_pqueue_remove_highest
   |<------------------
   |                chDbgCheckClassS
   |                 
   |                 _dbg_check_unlock
   |                 
   |                 _stats_stop_measure
   |                 _crit_thd
   |<------------------
   |                chDbgCheckClassS
   |                 
   |                 _dbg_check_unlock
   |                 
   |                 _stats_stop_measure
   |                 _crit_thd
   |<------------------
   |                chDbgCheckClassS
   |                 
   |                 _dbg_check_unlock
   |                 
   |                 _stats_stop_measure
   |                 _crit_thd
   |<------------------
   |                chDbgCheckClassS
   |                 
   |                 _dbg_check_unlock
   |                 
   |                 _stats_stop_measure
   |                 _crit_thd
   |<------------------
   |                chDbgCheckClassS
   |                 
   |                 _dbg_check_unlock
   |                 
   |                 _stats_stop_measure
   |                 _crit_thd
   |<------------------
   | ch_pqueue_remove_highest
```

7.24.4.15 chEvtWaitAll()

```
eventmask_t chEvtWaitAll (  
    eventmask_t events  )
```

Waits for all the specified events.

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

Parameters

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

Returns

The mask of the served and cleared events.
Function Class:

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

Here is the call graph for this function:

7.24.4.16 chEvtWaitOneTimeout()

```c
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 a higher priority.

Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>events</td>
<td>events that the function should wait for, ALL_EVENTS enables all the events</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 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:

7.24.4.17 chEvtWaitAnyTimeout()

eventmask_t chEvtWaitAnyTimeout(
    eventmask_t events,
    sysinterval_t timeout )

Waits for any of the specified events.

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

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>events</th>
<th>events that the function should wait for, ALL_EVENTS enables all the events</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>
7.24 Event Flags

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:

7.24.4.18 `chEvtWaitAllTimeout()`

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

Waits for all the specified events.

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

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th><code>events</code></th>
<th>events that the function should wait for, <code>ALL_EVENTS</code> requires 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>ChibiOS/RT</td>
<td></td>
<td>• <code>TIME_INFINITE</code> no timeout.</td>
</tr>
</tbody>
</table>
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:

7.24.4.19 chEvtObjectInit()

static void chEvtObjectInit (
    event_source_t * esp ) [inline], [static]

Initializes an Event Source.

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

Parameters
in 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.

7.24.4.20 chEvtRegisterMask()

static void chEvtRegisterMask (  
  event_source_t * esp,  
  event_listener_t * elp,  
  eventmask_t events ) [inline], [static]

Registers an Event Listener on an Event Source.

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

Note

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

Parameters

<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>events</td>
<td>the mask of events to be ORed to the thread when the event source is broadcasted</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:

```
chEvtRegisterMask  chEvtRegisterMaskWithFlags  chSysLock
chEvtRegisterMask  chEvtRegisterMaskWithFlags  chSysUnlock
```

7.24.4.21 chEvtRegister()

static void chEvtRegister (  
  event_source_t * esp,  
  event_listener_t * elp,  
  eventid_t event ) [inline], [static]

Registers an Event Listener on an Event Source.
Note

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

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>esp</th>
<th>pointer to the <code>event_source_t</code> structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>elp</td>
<td>pointer to the <code>event_listener_t</code> 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 <code>eventmask_t</code> 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.

Here is the call graph for this function:

![Call Graph](image)

7.24.422 chEvtIsListeningI()

```c
static bool chEvtIsListeningI ( event_source_t * esp ) { [inline], [static] }
```

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.
7.24 Event Flags

7.24.4.23 chEvtBroadcast()

static void chEvtBroadcast (event_source_t ∗ esp) [inline], [static]

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

Parameters

| in | esp | pointer to the event_source_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:

```
chEvtBroadcast
    \--- chEvtBroadcastFlags
          \--- chSchRescheduleS
          \--- chSysLock
          \--- chSysUnlock
```

7.24.4.24 chEvtBroadcastI()

static void chEvtBroadcastI (event_source_t ∗ esp) [inline], [static]

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

Postcondition

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

Parameters

| in | 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.

Here is the call graph for this function:

```
chEvtBroadcastI  chEvtBroadcastFlagsI  chDbgCheckClassI  chEvtSignalI
```

---

### 7.24.4.25 chEvtAddEventsI()

```c
static eventmask_t chEvtAddEventsI (
    eventmask_t events ) [inline], [static]
```

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

**Parameters**

- **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.

---

### 7.24.4.26 chEvtGetEventsX()

```c
static eventmask_t chEvtGetEventsX ( 
    void ) [inline], [static]
```

Returns the events mask.

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

**Returns**

The pending events mask.

Function Class:

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

7.25.1 Detailed Description

Synchronous inter-thread messages APIs and services.

Operation Mode

Synchronous messages are an easy to use and fast IPC mechanism, threads can both act as message servers and/or message clients, the mechanism allows data to be carried in both directions. Note that messages are not copied between the client and server threads but just a pointer passed so the exchange is very time efficient. Messages are scalar data types of type `msg_t` that are guaranteed to be size compatible with data pointers. Note that on some architectures function pointers can be larger that `msg_t`. Messages are usually processed in FIFO order but it is possible to process them in priority order by enabling the `CH_CFG_USE_MESSAGES_PRIORITY` option in `chconf.h`.

Precondition

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

Postcondition

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

Functions

- `msg_t chMsgSend (thread_t *tp, msg_t msg)`
  Sends a message to the specified thread.
- `thread_t * 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 * chMsgPoll (void)`
  Poll to check for an incoming message.
- `void chMsgRelease (thread_t *tp, msg_t msg)`
  Releases a sender thread specifying a response message.
- `static thread_t * chMsgWait (void)`
  Suspends the thread and waits for an incoming message.
- `static thread_t * chMsgWaitTimeout (sysinterval_t timeout)`
  Suspends the thread and waits for an incoming message or a timeout to occur.
- `static thread_t * chMsgPoll (void)`
  Poll to check for an incoming message.
- `static bool chMsgIsPending (thread_t *tp)`
  Evaluates to `true` if the thread has pending messages.
- `static msg_t chMsgGet (thread_t *tp)`
  Returns the message carried by the specified thread.
- `static void chMsgReleaseS (thread_t *tp, msg_t msg)`
  Releases the thread waiting on top of the messages queue.
7.25.2 Function Documentation

7.25.2.1 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>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>tp</code></td>
<td>the pointer to the thread</td>
</tr>
<tr>
<td><code>msg</code></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.

Here is the call graph for this function:

```
- chMsgSend
  - chSysLock
    - _dbg_check_lock
    - _stats_start_measure
      - _crit_thd
```

7.25.2.2 chMsgWaitS()

```c
thread_t * chMsgWaitS ( 
    void )
```

Suspends the thread and waits for an incoming message.
Postcondition

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

Note

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

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

Returns

A pointer to the thread carrying the message.

Function Class:

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

Here is the call graph for this function:

```
ch_queue_fifo_remove

chMsgIsPendingI

chSchGoSleepS

ch_pqueue_remove_highest

chDbgCheckClassS

chMsgWaitS

ch_queue_fifo_remove

chDbgCheckClassI

ch_pqueue_remove_highest

chMsgIsPendingI

chSchGoSleepS

chSysHalt

chMsgWaitS

ch_pqueue_remove_highest
```

### 7.25.2.3 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:

```
chMsgWaitTimeoutS
ch_queue_fifo_remove
chDbgCheckClassS
chMsgIsPendingI
chSchGoSleepTimeoutS
chSysHalt
chDbgCheckClassI
chVTDoSet
```

### 7.25.2.4 chMsgPollS()

`thread_t * chMsgPollS ( void )`

Poll to check for an incoming message.

**Postcondition**

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

**Note**

If the message is a pointer then you can assume that the data pointed by the message is stable until you invoke `chMsgRelease()` because the sending thread is suspended until then. The reference counter of the sender thread is not increased, the returned pointer is a temporary reference.

**Returns**

Result of the poll.
Return values

**NULL** if no incoming message waiting.

Function Class:

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

Here is the call graph for this function:

```
  chMsgPollS
  ch_queue_fifo_remove
  chMsgIsPendingI
  chDbgCheckClassI
```

### 7.25.2.5 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.

Here is the call graph for this function:

![Call Graph Image]

7.25.2.6  chMsgWait()

static thread_t* chMsgWait (  
    void ) [inline], [static]

Suspends the thread and waits for an incoming message.

Postcondition

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

Note

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

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

Returns

A pointer to the thread carrying the message.
Function Class:
Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

7.25.2.7 chMsgWaitTimeout()

static thread_t* chMsgWaitTimeout ( 
    sysinterval_t timeout ) [inline], [static]

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

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

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

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

<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_IMMEDIATE immediate timeout.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• TIME_INFINITE no timeout.</td>
</tr>
</tbody>
</table>

Returns

A pointer to the thread carrying the message.

Return values

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:

```
chMsgWaitTimeout
chMsgWaitTimeoutS
chSysLock
chSysUnlock
ch_queue_fifo_remove
chDbgCheckClassSchMsgIsPendingI
chSchGoSleepTimeoutS
_dbg_check_lock
_stats_start_measure
crthd
_dbg_check_unlock
_stats_stop_measure
crthd
```

7.25.2.8 chMsgPoll()

static thread_t* chMsgPoll ( void ) [inline], [static]

Poll to check for an incoming message.
Postcondition

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

Note

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

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

Returns

A pointer to the thread carrying the message.

Return values

- **NULL** if no incoming message waiting.

Function Class:

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

Here is the call graph for this function:
7.25.2.9 chMsgIsPendingI()

static bool chMsgIsPendingI (  
    thread_t * tp ) [inline], [static]

Evaluates to true if the thread has pending messages.

Parameters

|    | tp | pointer to the thread |

Returns

The pending messages status.

Function Class:

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

Here is the call graph for this function:

```
    chMsgIsPendingI  -->  chDbgCheckClassI  -->  chSysHalt
```

7.25.2.10 chMsgGet()

static msg_t chMsgGet (  
    thread_t * tp ) [inline], [static]

Returns the message carried by the specified thread.

Precondition

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

Parameters

|    | tp | pointer to the thread |
7.25 Synchronous Messages

Returns

The message carried by the sender.

Function Class:

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

7.25.2.11 chMsgReleaseS()

static void chMsgReleaseS (  
        thread_t * tp,  
        msg_t msg ) [inline], [static]

Releases the thread waiting on top of the messages queue.

Precondition

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

Parameters

| in  | 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.

Here is the call graph for this function:
7.26 Dynamic Threads

7.26.1 Detailed Description

Dynamic threads related APIs and services.

Functions

- thread_t * chThdCreateFromHeap (memory_heap_t *heapp, size_t size, const char *name, tprio_t prio, tfunc_t pf, void *arg)
  
  Creates a new thread allocating the memory from the heap.

- thread_t * chThdCreateFromMemoryPool (memory_pool_t *mp, const char *name, tprio_t prio, tfunc_t pf, void *arg)
  
  Creates a new thread allocating the memory from the specified memory pool.

7.26.2 Function Documentation

7.26.2.1 chThdCreateFromHeap()

thread_t * chThdCreateFromHeap {
    memory_heap_t * heapp,
    size_t size,
    const char * name,
    tprio_t prio,
    tfunc_t pf,
    void * arg
}

Creates a new thread allocating the memory from the heap.

Precondition

The configuration options CH_CFG_USE_DYNAMIC and CH_CFG_USE_HEAP must be enabled in order to use this function.

Note

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

The memory allocated for the thread is not released automatically; it is responsibility of the creator thread to call chThdWait() and then release the allocated memory.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>heapp</strong></td>
<td>heap from which allocate the memory or NULL for the default heap</td>
</tr>
<tr>
<td><strong>size</strong></td>
<td>size of the working area to be allocated</td>
</tr>
<tr>
<td><strong>name</strong></td>
<td>thread name</td>
</tr>
<tr>
<td><strong>prio</strong></td>
<td>the priority level for the new thread</td>
</tr>
<tr>
<td><strong>pf</strong></td>
<td>the thread function</td>
</tr>
<tr>
<td><strong>arg</strong></td>
<td>an argument passed to the thread function. It can be NULL.</td>
</tr>
</tbody>
</table>
7.26 Dynamic Threads

Returns

The pointer to the thread_t structure allocated for the thread into the working space area.

Return values

NULL if the memory cannot be allocated.

Function Class:

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

Here is the call graph for this function:

7.26.2.2 chThdCreateFromMemoryPool()

thread_t * chThdCreateFromMemoryPool (  
    memory_pool_t * mp,  
    const char * name,  
    tprio_t prio,  
    tfunc_t pf,  
    void * arg )

Creates a new thread allocating the memory from the specified memory pool.

Precondition

The configuration options CH_CFG_USE_DYNAMIC and CH_CFG_USE_MEMPOOLS must be enabled in order to use this function.

The pool must be initialized to contain only objects with alignment PORT_WORKING_AREA_ALIGN.
Note

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

The memory allocated for the thread is not released automatically, it is responsibility of the creator thread to call `chThdWait()` and then release the allocated memory.

Parameters

| in | mp | pointer to the memory pool object |
| in | name | thread name |
| in | prio | the priority level for the new thread |
| in | pf | the thread function |
| in | arg | an argument passed to the thread function. It can be `NULL`. |

Returns

The pointer to the `thread_t` structure allocated for the thread into the working space area.

Return values

- `NULL` if the memory pool is empty.

Function Class:

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

Here is the call graph for this function:
7.27 Registry

7.27.1 Detailed Description

Threads Registry related APIs and services.

Operation mode

The Threads Registry is a double linked list that holds all the active threads in the system. Operations defined for the registry:

- **First**, returns the first, in creation order, active thread in the system.
- **Next**, returns the next, in creation order, active thread in the system.

The registry is meant to be mainly a debug feature, for example, using the registry a debugger can enumerate the active threads in any given moment or the shell can print the active threads and their state.

Another possible use is for centralized threads memory management, terminating threads can pulse an event source and an event handler can perform a scansion of the registry in order to recover the memory.

Precondition

In order to use the threads registry the **CH_CFG_USE_REGISTRY** option must be enabled in `chconf.h`.

Macros

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

Data Structures

- **struct chdebug_t**
  
  ChibiOS/RT memory signature record.

Functions

- **thread_t * chRegFirstThread (void)**
  
  Returns the first thread in the system.
- **thread_t * chRegNextThread (thread_t *tp)**
  
  Returns the thread next to the specified one.
- **thread_t * chRegFindThreadByName (const char *name)**
  
  Retrieves a thread pointer by name.
- **thread_t * chRegFindThreadByPointer (thread_t *tp)**
  
  Confirms that a pointer is a valid thread pointer.
- **thread_t * chRegFindThreadByWorkingArea (stkalign_t *wa)**
  
  Confirms that a working area is being used by some active thread.
- **static void chRegSetThreadName (const char *name)**
  
  Sets the current thread name.
- **static const char * chRegGetThreadNameX (thread_t *tp)**
  
  Returns the name of the specified thread.
- **static void chRegSetThreadNameX (thread_t *tp, const char *name)**
  
  Changes the name of the specified thread.
### 7.27.2 Macro Definition Documentation

#### 7.27.2.1 REG_REMOVE

```c
#define REG_REMOVE(
  tp )
```

**Value:**

```c
{        \                   \       
  (tp)->older->newer = (tp)->newer; \ 
  (tp)->newer->older = (tp)->older; \ 
}
```

Removes a thread from the registry list.

**Note**

This macro is not meant for use in application code.

**Parameters**

- `tp` thread to remove from the registry

#### 7.27.2.2 REG_INSERT

```c
#define REG_INSERT(
  tp )
```

**Value:**

```c
{        \                   \       
  (tp)->newer = (thread_t *)&ch.rlist; \ 
  (tp)->older = ch.rlist.older; \       \ 
  (tp)->older->newer = (tp); \          \ 
  ch.rlist.older = (tp); \               \ 
}
```

Adds a thread to the registry list.

**Note**

This macro is not meant for use in application code.

**Parameters**

- `tp` thread to add to the registry
7.27 Registry

7.27.3 Function Documentation

7.27.3.1 chRegFirstThread()

```c
thread_t * chRegFirstThread ( void )
```

Returns the first thread in the system.
Returns the most ancient thread in the system, usually this is the main thread unless it terminated. A reference is added to the returned thread in order to make sure its status is not lost.

Note

This function cannot return NULL because there is always at least one thread in the system.

Returns

A reference to the most ancient thread.

Function Class:

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

Here is the call graph for this function:

![Call Graph](call_graph.png)

7.27.3.2 chRegNextThread()

```c
thread_t * chRegNextThread ( thread_t * tp )
```

Returns the thread next to the specified one.

The reference counter of the specified thread is decremented and the reference counter of the returned thread is incremented.
Parameters

\[ \text{in} \quad \mathit{tp} \quad \text{pointer to the thread} \]

Returns

A reference to the next thread.

Return values

\[ \text{NULL} \quad \text{if there is no next thread.} \]

Function Class:

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

Here is the call graph for this function:

```
chRegNextThread
chSysLock
chSysUnlock
chThdRelease
_dbg_check_lock
_stats_start_measure
_crit_thd
_dbg_check_unlock
_stats_stop_measure
_crit_thd
chHeapFree
chPoolFree
chThdGetWorkingAreaX
```

7.27.3.3 \texttt{chRegFindThreadByName()}

\texttt{thread_t * chRegFindThreadByName ( const char * name )}

Retrieves a thread pointer by name.

Note

The reference counter of the found thread is increased by one so it cannot be disposed incidentally after the pointer has been returned.

Parameters

\[ \text{in} \quad \mathit{name} \quad \text{the thread name} \]
7.27 Registry

Returns
A pointer to the found thread.

Return values

| NULL         | if a matching thread has not been found. |

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

Here is the call graph for this function:

![Call Graph](image)

7.27.3.4 chRegFindThreadByPointer()

```c
thread_t * chRegFindThreadByPointer ( thread_t * tp )
```

Confirms that a pointer is a valid thread pointer.

Note
The reference counter of the found thread is increased by one so it cannot be disposed incidentally after the pointer has been returned.

Parameters

|    | tp | pointer to the thread |

Returns
A pointer to the found thread.
Return values

| NULL | if a matching thread has not been found. |

Function Class:

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

Here is the call graph for this function:

![Call Graph](image)

### 7.27.3.5 chRegFindThreadByWorkingArea()

```c
thread_t * chRegFindThreadByWorkingArea ( 
  stkalign_t * wa )
```

Confirms that a working area is being used by some active thread.

Note

The reference counter of the found thread is increased by one so it cannot be disposed incidentally after the pointer has been returned.

Parameters

| in  | wa   | pointer to a static working area |

Returns

A pointer to the found thread.

Return values

| NULL | if a matching thread has not been found. |
Function Class:
Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

```
chRegFindThreadByWorkingArea
chRegFirstThread
chRegNextThread
chThdGetWorkingAreaX
chSysLock
chSysUnlock
chThdRelease
```

7.27.3.6 chRegSetThreadName()

```c
static void chRegSetThreadName (const char * name) [inline], [static]
```

Sets the current thread name.

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

Parameters

| in  | name  | thread name as a zero terminated string |

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

7.27.3.7 chRegGetThreadNameX()

```c
static const char* chRegGetThreadNameX (thread_t * tp ) [inline], [static]
```

Returns the name of the specified thread.

Precondition
This function only returns the pointer to the name if the option CH_CFG_USE_REGISTRY is enabled else NULL is returned.
Parameters

| in | tp | pointer to the thread |

Returns

Thread name as a zero terminated string.

Return values

| NULL | if the thread name has not been set. |

### 7.27.3.8 chRegSetThreadNameX()

```c
static void chRegSetThreadNameX (  
    thread_t * tp,  
    const char * name ) [inline], [static]
```

Changes the name of the specified thread.

Precondition

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

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>tp</th>
<th>pointer to the thread</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>name</td>
<td>thread name as a zero terminated string</td>
</tr>
</tbody>
</table>

Function Class:

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

7.28.1 Detailed Description

Modules

- Checks and Assertions
- Tracing
- Statistics
7.29 Checks and Assertions

7.29.1 Detailed Description

Debug APIs and services:

• Runtime system state and call protocol check. The following panic messages can be generated:

  – SV#1, misplaced `chSysDisable()`.
    • Called from an ISR.
    • Called from a critical zone.
  – SV#2, misplaced `chSysSuspend()`
    • Called from an ISR.
    • Called from a critical zone.
  – SV#3, misplaced `chSysEnable()`.
    • Called from an ISR.
    • Called from a critical zone.
  – SV#4, misplaced `chSysLock()`.
    • Called from an ISR.
    • Called from a critical zone.
  – SV#5, misplaced `chSysUnlock()`.
    • Called from an ISR.
    • Not called from a critical zone.
  – SV#6, misplaced `chSysLockFromISR()`.
    • Not called from an ISR.
    • Called from a critical zone.
  – SV#7, misplaced `chSysUnlockFromISR()`.
    • Not called from an ISR.
    • Not called from a critical zone.
  – SV#8, misplaced `CH_IRQ_PROLOGUE()`.
    • Not called at ISR begin.
    • Called from a critical zone.
  – SV#9, misplaced `CH_IRQ_EPILOGUE()`.
    • `CH_IRQ_PROLOGUE()` missing.
    • Not called at ISR end.
    • Called from a critical zone.
  – SV#10, misplaced I-class function.
    • I-class function not called from within a critical zone.
  – SV#11, misplaced S-class function.
    • S-class function not called from within a critical zone.
    • Called from an ISR.

• Parameters check.

• Kernel assertions.

Note

Stack checks are not implemented in this module but in the port layer in an architecture-dependent way.
7.29 Checks and Assertions

Debug related settings

- #define CH_DBG_STACK_FILL_VALUE 0x55
  Fill value for thread stack area in debug mode.

Macro Functions

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

Functions

- void _dbg_check_disable (void)
  Guard code for chSysDisable().
- void _dbg_check_suspend (void)
  Guard code for chSysSuspend().
- void _dbg_check_enable (void)
  Guard code for chSysEnable().
- void _dbg_check_lock (void)
  Guard code for chSysLock().
- void _dbg_check_unlock (void)
  Guard code for chSysUnlock().
- void _dbg_check_lock_from_isr (void)
  Guard code for chSysLockFromIsr().
- void _dbg_check_unlock_from_isr (void)
  Guard code for chSysUnlockFromIsr().
- void _dbg_check_enter_isr (void)
  Guard code for CH_IRQ_PROLOGUE().
- void _dbg_check_leave_isr (void)
  Guard code for CH_IRQ_EPILOGUE().
- void chDbgCheckClassI (void)
  I-class functions context check.
- void chDbgCheckClassS (void)
  S-class functions context check.

7.29.2 Macro Definition Documentation

7.29.2.1 CH_DBG_STACK_FILL_VALUE

#define CH_DBG_STACK_FILL_VALUE 0x55
Fill value for thread stack area in debug mode.
7.29.2.2 chDbgCheck

#define chDbgCheck(
  c )

Value:

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

Function parameters check.

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

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

Parameters

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

Function Class:

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

7.29.2.3 chDbgAssert

#define chDbgAssert(
  c ,
  r )

Value:

/*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

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>c</code></td>
<td><code>c</code></td>
<td>the condition to be verified to be true</td>
</tr>
<tr>
<td><code>r</code></td>
<td><code>r</code></td>
<td>a remark string</td>
</tr>
</tbody>
</table>

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

7.29.3 Function Documentation

7.29.3.1 _dbg_check_disable()

```c
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:

```
_dbg_check_disable -> chSysHalt -> _trace_halt
```

7.29.3.2 _dbg_check_suspend()

```c
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]

7.29.3.3  _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]

7.29.3.4  _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:

```
_dbg_check_lock → chSysHalt → _trace_halt
```

7.29.3.5 \_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:

```
_dbg_check_unlock → chSysHalt → _trace_halt
```

7.29.3.6 \_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:

```
_dbg_check_lock_from_isr  chSysHalt  _trace_halt
```

### 7.29.3.7  _dbg_check_unlock_from_isr()

```c
void _dbg_check_unlock_from_isr (  
    void  
)
```

Guard code for `chSysUnlockFrom ISR()`.

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

Here is the call graph for this function:

```
_dbg_check_unlock_from_isr  chSysHalt  _trace_halt
```

### 7.29.3.8  _dbg_check_enter_isr()

```c
void _dbg_check_enter_isr (  
    void  
)
```

Guard code for `CH_IRQ_PROLOGUE()`. 
Function Class:
Not an API, this function is for internal use only.

Here is the call graph for this function:

7.29.3.9 _dbg_check_leave_isr()

void _dbg_check_leave_isr (  
  void  )

Guard code for CH_IRQ_EPILOGUE().

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

Here is the call graph for this function:

7.29.3.10 chDbgCheckClassI()

void chDbgCheckClassI (  
  void  )

I-class functions context check.

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

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

Here is the call graph for this function:

![Call Graph]

### 7.29.3.11 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:

![Call Graph]
7.30 Tracing

7.30.1 Detailed Description

System events tracing service.

Trace record types

- \#define CH_TRACE_TYPE_UNUSED 0U
- \#define CH_TRACE_TYPE_SWITCH 1U
- \#define CH_TRACE_TYPE_ISR_ENTER 2U
- \#define CH_TRACE_TYPE_ISR_LEAVE 3U
- \#define CH_TRACE_TYPE_HALT 4U
- \#define CH_TRACE_TYPE_USER 5U

Events to trace

- \#define CH_DBG_TRACE_MASK_DISABLED 255U
- \#define CH_DBG_TRACE_MASK_NONE 0U
- \#define CH_DBG_TRACE_MASK_SWITCH 1U
- \#define CH_DBG_TRACE_MASK_ISR 2U
- \#define CH_DBG_TRACE_MASK_HALT 4U
- \#define CH_DBG_TRACE_MASK_USER 8U
- \#define CH_DBG_TRACE_MASK_SLOW
- \#define CH_DBG_TRACE_MASK_ALL

Debug related settings

- \#define CH_DBG_TRACE_MASK CH_DBG_TRACE_MASK_DISABLED

  Trace buffer entries.
- \#define CH_DBG_TRACE_BUFFER_SIZE 128

  Trace buffer entries.

Data Structures

- struct ch_trace_event_t

  Trace buffer record.
- struct ch_trace_buffer_t

  Trace buffer header.
Functions

- static NOINLINE void trace_next (void)
  
  Writes a time stamp and increases the trace buffer pointer.
- void _trace_init (void)
  
  Trace circular buffer subsystem initialization.
- void _trace_switch (thread_t *ntp, thread_t *otp)
  
  Inserts in the circular debug trace buffer a context switch record.
- void _trace_isr_enter (const char *isr)
  
  Inserts in the circular debug trace buffer an ISR-enter record.
- void _trace_isr_leave (const char *isr)
  
  Inserts in the circular debug trace buffer an ISR-leave record.
- void _trace_halt (const char *reason)
  
  Inserts in the circular debug trace buffer an halt record.
- void chDbgWriteTraceI (void *up1, void *up2)
  
  Adds an user trace record to the trace buffer.
- void chDbgWriteTrace (void *up1, void *up2)
  
  Adds an user trace record to the trace buffer.
- void chDbgSuspendTraceI (uint16_t mask)
  
  Suspends one or more trace events.
- void chDbgSuspendTrace (uint16_t mask)
  
  Suspends one or more trace events.
- void chDbgResumeTraceI (uint16_t mask)
  
  Resumes one or more trace events.
- void chDbgResumeTrace (uint16_t mask)
  
  Resumes one or more trace events.

7.30.2 Macro Definition Documentation

7.30.2.1 CH_DBG_TRACE_MASK

#define CH_DBG_TRACE_MASK CH_DBG_TRACE_MASK_DISABLED

Trace buffer entries.

7.30.2.2 CH_DBG_TRACE_BUFFER_SIZE

#define CH_DBG_TRACE_BUFFER_SIZE 128

Trace buffer entries.

Note

The trace buffer is only allocated if CH_DBG_TRACE_MASK is different from CH_DBG_TRACE_MASK_DISABLED.
7.30.3 Function Documentation

7.30.3.1 trace_next()

static NOINLINE void trace_next (    
    void ) [static]

Writes a time stamp and increases the trace buffer pointer.

Function Class:

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

Here is the call graph for this function:

```
trace_next \rightarrow chVTGetSystemTimeX
```

7.30.3.2 _trace_init()

void _trace_init (    
    void )

Trace circular buffer subsystem initialization.

Note

Internal use only.

7.30.3.3 _trace_switch()

void _trace_switch (    
    thread_t * ntp,    
    thread_t * otp )

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

<table>
<thead>
<tr>
<th></th>
<th>ntp</th>
<th>the thread being switched in</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>otp</td>
<td>the thread being switched out</td>
</tr>
</tbody>
</table>

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

7.30.3.4 _trace_isr_enter()

```c
void _trace_isr_enter ( 
    const char * isr )
```

Inserts in the circular debug trace buffer an ISR-enter record.

Parameters

|   | isr   | name of the isr |

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

7.30.3.5 _trace_isr_leave()

```c
void _trace_isr_leave ( 
    const char * isr )
```

Inserts in the circular debug trace buffer an ISR-leave record.

Parameters

|   | isr   | name of the isr |

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

7.30.3.6 _trace_halt()

```c
void _trace_halt ( 
    const char * reason )
```
7.30 Tracing

Inserts in the circular debug trace buffer an halt record.

Parameters

| in  | reason | the halt error string |

Function Class:

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

### 7.30.3.7 chDbgWriteTraceI()

```c
void chDbgWriteTraceI (  
    void * up1,  
    void * up2  )
```

Adds an user trace record to the trace buffer.

Parameters

| in  | up1    | user parameter 1 |
| in  | up2    | user parameter 2 |

Function Class:

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

Here is the call graph for this function:

![Call Graph]

#### 7.30.3.8 chDbgWriteTrace()

```c
void chDbgWriteTrace (  
    void * up1,  
    void * up2  )
```

Adds an user trace record to the trace buffer.
7.30.3.9 chDbgSuspendTraceI()

void chDbgSuspendTraceI (  
    uint16_t mask )  

Suspends one or more trace events.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>mask</th>
<th>mask of the trace events to be suspended</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:
Function 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](attachment:call_graph.png)

7.30.3.10 **chDbgSuspendTrace()**

```c
void chDbgSuspendTrace ( uint16_t mask )
```

Suspends one or more trace events.

**Parameters**

| in | mask | mask of the trace events to be suspended |

**Function Class:**

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

Here is the call graph for this function:

![Call Graph](attachment:call_graph2.png)
7.30.3.11 chDbgResumeTraceI()

```c
void chDbgResumeTraceI (    
    uint16_t  mask )
```

Resumes one or more trace events.

**Parameters**

| in | mask | mask of the trace events to be resumed |

**Function Class:**

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

Here is the call graph for this function:

![Call Graph](chDbgResumeTraceI chDbgCheckClassI chSysHalt)

7.30.3.12 chDbgResumeTrace()

```c
void chDbgResumeTrace (    
    uint16_t  mask )
```

Resumes one or more trace events.

**Parameters**

| in | mask | mask of the trace events to be resumed |
Function Class:

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

Here is the call graph for this function:
7.31 Statistics

7.31.1 Detailed Description

Statistics services.

Data Structures

• struct kernel_stats_t
  Type of a kernel statistics structure.

Functions

• void _stats_init (void)
  Initializes the statistics module.
• void _stats_increase_irq (void)
  Increases the IRQ counter.
• void _stats_ctxswc (thread_t *ntp, thread_t *otp)
  Updates context switch related statistics.
• void _stats_start_measure_crit_thd (void)
  Starts the measurement of a thread critical zone.
• void _stats_stop_measure_crit_thd (void)
  Stops the measurement of a thread critical zone.
• void _stats_start_measure_crit_isr (void)
  Starts the measurement of an ISR critical zone.
• void _stats_stop_measure_crit_isr (void)
  Stops the measurement of an ISR critical zone.

7.31.2 Function Documentation

7.31.2.1 _stats_init()

void _stats_init (void)
  Initializes the statistics module.

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)
7.31.2.2 _stats_increase_irq()

```c
void _stats_increase_irq (void)
```

Increases the IRQ counter.

7.31.2.3 _stats_ctxswc()

```c
void _stats_ctxswc (thread_t * ntp,
                   thread_t * otp)
```

Updates context switch related statistics.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>ntp</th>
<th>the thread to be switched in</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>otp</td>
<td>the thread to be switched out</td>
</tr>
</tbody>
</table>

Here is the call graph for this function:

```
_stats_ctxswc  \rightarrow chTMChainMeasurementToX
```

7.31.2.4 _stats_start_measure_crit_thd()

```c
void _stats_start_measure_crit_thd (void)
```

Starts the measurement of a thread critical zone.

Here is the call graph for this function:

```
_stats_start_measure_crit_thd  \rightarrow chTMStartMeasurementX
```
## 7.31.2.5 _stats_stop_measure_crit_thd()

```c
void _stats_stop_measure_crit_thd (
    void )
```

Stops the measurement of a thread critical zone.

Here is the call graph for this function:

![Call Graph](call_graph.png)

## 7.31.2.6 _stats_start_measure_crit_isr()

```c
void _stats_start_measure_crit_isr (
    void )
```

Starts the measurement of an ISR critical zone.

Here is the call graph for this function:

![Call Graph](call_graph.png)
7.31.2.7  _stats_stop_measure_crit_isr()

void _stats_stop_measure_crit_isr (  
    void  )

Stops the measurement of an ISR critical zone.

Here is the call graph for this function:
7.32 OS Library

7.32.1 Detailed Description

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

Modules

- Version Numbers and Identification
- Synchronization
- Memory Management
- Complex Services
7.33 Version Numbers and Identification

7.33.1 Detailed Description

OS Library related info.

Macros

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

ChibiOS/LIB version identification

- `#define CH_OSLIB_VERSION "1.2.1"
  OS Library version string.
- `#define CH_OSLIB_MAJOR 1`
  OS Library version major number.
- `#define CH_OSLIB_MINOR 2`
  OS Library version minor number.
- `#define CH_OSLIB_PATCH 1`
  OS Library version patch number.

Functions

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

7.33.2 Macro Definition Documentation

7.33.2.1 _CHIBIOS_OSLIB_

```
#define _CHIBIOS_OSLIB_
```

ChibiOS/LIB identification macro.

7.33.2.2 CH_OSLIB_STABLE

```
#define CH_OSLIB_STABLE 1
```

Stable release flag.
7.33.2.3 CH_OSLIB_VERSION

#define CH_OSLIB_VERSION "1.2.1"

OS Library version string.

7.33.2.4 CH_OSLIB_MAJOR

#define CH_OSLIB_MAJOR 1

OS Library version major number.

7.33.2.5 CH_OSLIB_MINOR

#define CH_OSLIB_MINOR 2

OS Library version minor number.

7.33.2.6 CH_OSLIB_PATCH

#define CH_OSLIB_PATCH 1

OS Library version patch number.

7.33.3 Function Documentation
7.33.3.1 _oslib_init()

```c
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:

![Call Graph](image_url)
7.34 Synchronization

7.34.1 Detailed Description

Synchronization services.

Modules

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

7.35.1 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.

7.35.2 Macro Definition Documentation
7.35.2.1 _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>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>name</td>
</tr>
<tr>
<td>in</td>
<td>taken</td>
</tr>
</tbody>
</table>

7.35.2.2 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>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>name</td>
</tr>
<tr>
<td>in</td>
<td>taken</td>
</tr>
</tbody>
</table>

7.35.3 Typedef Documentation

7.35.3.1 binary_semaphore_t

typedef struct ch_binary_semaphore binary_semaphore_t

Binary semaphore type.

7.35.4 Function Documentation
### 7.35 Binary Semaphores

#### 7.35.4.1 chBSemObjectInit()

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

Initializes a binary semaphore.

**Parameters**

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

Here is the call graph for this function:

![Call Graph](https://chart.png)

#### 7.35.4.2 chBSemWait()

```c
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>Message</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSG_OK</td>
<td>if the binary semaphore has been successfully taken.</td>
</tr>
<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.

Here is the call graph for this function:

```
chBSemWaitS  
<table>
<thead>
<tr>
<th></th>
<th>chSemWaitS</th>
<th>chSysLock</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>chBSemWait</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

7.35.4.3 chBSemWaitS()

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

Wait operation on the binary semaphore.

Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td><code>bsp</code> pointer to a <code>binary_semaphore_t</code> structure</td>
</tr>
</tbody>
</table>

Returns

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

Return values

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

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
  \-- chSemWaitS
  \-- ch_queue_isempty
      \-- ch_queue_notempty
```

7.35.4.4 chBSemWaitTimeoutS()

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

Wait operation on the binary semaphore.

### Parameters

<table>
<thead>
<tr>
<th>in</th>
<th><strong>bsp</strong></th>
<th>pointer to a binary_semaphore_t structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<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></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

| **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:

![Call Graph](image-url)

### 7.35.4.5 `chBSemWaitTimeout()`

```c
static msg_t chBSemWaitTimeout ( 
    binary_semaphore_t * bsp, 
    sysinterval_t timeout ) {inline}, [static]
```

Wait operation on the binary semaphore.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>bsp</th>
<th>pointer to a <code>binary_semaphore_t</code> 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>• <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**

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 <code>bsemReset()</code></td>
</tr>
<tr>
<td>MSG_TIMEOUT</td>
<td>if the binary semaphore has not been signaled or reset within the specified timeout.</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:

```
<table>
<thead>
<tr>
<th></th>
<th>chBSemWaitTimeout</th>
<th>chSemWaitTimeout</th>
<th>chSemWaitTimeoutS</th>
</tr>
</thead>
<tbody>
<tr>
<td>chBSemWaitTimeout</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>chSemWaitTimeout</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>chSemWaitTimeoutS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>chSysLock</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>chSysUnlock</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

### 7.35.4.6 chBSemResetI()

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

Reset operation on the binary semaphore.

**Note**

The released threads can recognize they were waked up by a reset rather than a signal because the `bsemWait()` 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>
<tr>
<td></td>
<td></td>
<td>• <code>false</code>, the new state is not taken.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <code>true</code>, the new state is taken.</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)

### 7.35.4.7 chBSemReset()

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

Here is the call graph for this function:

![Call Graph](image)
7.35 Binary Semaphores

7.35.4.8 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:

```
chBSemSignal
  chDbgCheckClassI
  chSemSignalI
  chSysHalt
  ch_queue_fifo_remove
  ch_queue_isempty
  ch_queue_notempty
  chSchReadyI
```

7.35.4.9 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:

7.35.4.10 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:
7.36 Mailboxes

7.36.1 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.

7.36.2 Macro Definition Documentation
7.36.2.1 _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>in</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>

7.36.2.2 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>in</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>

7.36.3 Function Documentation
7.36.3.1 chMBObjectInit()

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

Initializes a `mailbox_t` object.

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>mbp</td>
<td>the pointer to the <code>mailbox_t</code> structure to be initialized</td>
</tr>
<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.

Here is the call graph for this function:

```
chMBObjectInit      chThdQueueObjectInit     ch_queue_init
```

7.36.3.2 chMBReset()

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

Resets a `mailbox_t` object.

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

**Postcondition**

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

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>mbp</td>
<td>the pointer to an initialized <code>mailbox_t</code> object</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)

### 7.36.3.3 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:

```
    chMBResetI
    chDbgCheckClassI
    chThdDequeueAllI
    chSysHalt
    ch_queue_notempty
    chThdDoDequeueNextI
```

### 7.36.3.4 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**

<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>
<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 posted.</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:**

Normal API, 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 Diagram]

7.36.3.5 chMBPostTimeoutS()

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

Posts a message into a mailbox.

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

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>mbp</th>
<th>the pointer to an initialized <code>mailbox_t</code> object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>msg</td>
<td>the message to be posted on the mailbox</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>• <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 operation status.
Return values

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSG_OK</td>
<td>if a message has been correctly posted.</td>
</tr>
<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)

### 7.36.3.6 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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in mbp</td>
<td>the pointer to an initialized mailbox_t object</td>
</tr>
<tr>
<td>in msg</td>
<td>the message to be posted on the mailbox</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>MSG_OK</td>
<td>if a message has been correctly posted.</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 full and the message cannot be posted.</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]

7.36.3.7 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>in mbp</td>
<td>the pointer to an initialized <code>mailbox_t</code> object</td>
</tr>
<tr>
<td>in msg</td>
<td>the message to be posted on the mailbox</td>
</tr>
<tr>
<td>in timeout</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>MSG_OK</th>
<th>if a message has been correctly posted.</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:

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

Here is the call graph for this function:

```
chMBPostAheadTimeout
chMBPostAheadTimeoutS
chSysLock
chSysUnlock
chDbgCheckClass
chMBGetFreeCount
chSchReschedule
chThdDequeueNext
chThdEnqueueTimeout
_dbg_check_lock
_stats_start_measure
_crit_thd
_dbg_check_unlock
_stats_stop_measure
_crit_thd
```

### 7.36.3.8 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>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>
<tr>
<td></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</td>
<td>immediate timeout</td>
</tr>
<tr>
<td></td>
<td>TIME_INFINITE</td>
<td>no timeout</td>
</tr>
</tbody>
</table>

ChibiOS
Returns

The operation status.

Return values

<table>
<thead>
<tr>
<th>MSG_OK</th>
<th>if a message has been correctly posted.</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:

7.36.3.9 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 <code>mailbox_t</code> 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:

![Call Graph](image)

7.36.3.10 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

| in | mbp | the pointer to an initialized mailbox_t object |
| out | msgp | pointer to a message variable for the received message |
| 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>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 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:

```
7.36.3.11 chMBFetchTimeoutS()

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

Retrieves a message from a mailbox.

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

Parameters

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

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

Here is the call graph for this function:

7.36.3.12 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

<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>
</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 mailbox is empty and a message cannot be fetched. |

Function Class:

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

Here is the call graph for this function:

```
7.36.3.13 chMBGetSizeI()

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

Returns the mailbox buffer size as number of messages.

Parameters

| in  | 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.

7.36.3.14  chMBGetUsedCountI()

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

Returns the number of used message slots into a mailbox.

Parameters

in  mbp  the pointer to an initialized mailbox_t object

Returns

The number of queued messages.

Function Class:

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

Here is the call graph for this function:

chMBGetUsedCountI chDbgCheckClassI chSysHalt

7.36.3.15  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](image)

### 7.36.3.16 chMBPeekI()

```c
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
```

### 7.36.3.17 chMBResumeX()

```c
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.
7.37 Pipes

7.37.1 Detailed Description

Macros

- \#define _PIPE_DATA(name, buffer, size)
  
  Data part of a static pipe initializer.

- \#define PIPE_DECL(name, buffer, size) pipe_t name = _PIPE_DATA(name, buffer, size)
  
  Static pipe initializer.

Data Structures

- struct pipe_t
  
  Structure representing a pipe object.

Functions

- static size_t pipe_write (pipe_t *pp, const uint8_t *bp, size_t n)
  
  Non-blocking pipe write.

- static size_t pipe_read (pipe_t *pp, uint8_t *bp, size_t n)
  
  Non-blocking pipe read.

- void chPipeObjectInit (pipe_t *pp, uint8_t *buf, size_t n)
  
  Initializes a mailbox_t object.

- void chPipeReset (pipe_t *pp)
  
  Resets a pipe_t object.

- size_t chPipeWriteTimeout (pipe_t *pp, const uint8_t *bp, size_t n, sysinterval_t timeout)
  
  Pipe write with timeout.

- size_t chPipeReadTimeout (pipe_t *pp, uint8_t *bp, size_t n, sysinterval_t timeout)
  
  Pipe read with timeout.

- static size_t chPipeGetSize (const pipe_t *pp)
  
  Returns the pipe buffer size as number of bytes.

- static size_t chPipeGetUsedCount (const pipe_t *pp)
  
  Returns the number of used byte slots into a pipe.

- static size_t chPipeGetFreeCount (const pipe_t *pp)
  
  Returns the number of free byte slots into a pipe.

- static void chPipeResume (pipe_t *pp)
  
  Terminates the reset state.

7.37.2 Macro Definition Documentation
7.37 Pipes

7.37.2.1 _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>

7.37.2.2 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>

7.37.3 Function Documentation
### 7.37.3.1 pipe_write()

```c
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>In</th>
<th>pp</th>
<th>the pointer to an initialized pipe_t object</th>
</tr>
</thead>
<tbody>
<tr>
<td>In</td>
<td>bp</td>
<td>pointer to the data buffer</td>
</tr>
<tr>
<td>In</td>
<td>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.

### 7.37.3.2 pipe_read()

```c
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>In</th>
<th>pp</th>
<th>the pointer to an initialized pipe_t object</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>bp</td>
<td>pointer to the data buffer</td>
</tr>
<tr>
<td>In</td>
<td>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.

### 7.37.3.3 chPipeObjectInit()

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

Initializes a `mailbox_t` object.

**Parameters**

<table>
<thead>
<tr>
<th>out</th>
<th>pp</th>
<th>the pointer to the <code>pipe_t</code> structure to be initialized</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>buf</td>
<td>pointer to the pipe buffer as an array of <code>uint8_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.

### 7.37.3.4 chPipeReset()

```c
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.
### 7.37.3.5 chPipeWriteTimeout()

```c
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**

<table>
<thead>
<tr>
<th>in</th>
<th>pp</th>
<th>the pointer to an initialized <code>pipe_t</code> object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>bp</td>
<td>pointer to the data buffer</td>
</tr>
<tr>
<td>in</td>
<td>n</td>
<td>the number of bytes to be written, the value 0 is reserved</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>• <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 number of bytes effectively transferred. A number lower than `n` means that a timeout occurred or the pipe went in reset state.

**Function Class:**

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

### 7.37.3.6 chPipeReadTimeout()

```c
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**

<table>
<thead>
<tr>
<th>in</th>
<th>pp</th>
<th>the pointer to an initialized <code>pipe_t</code> object</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>bp</td>
<td>pointer to the data buffer</td>
</tr>
<tr>
<td>in</td>
<td>n</td>
<td>the number of bytes to be read, the value 0 is reserved</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>• <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 number of bytes effectively transferred. A number lower than \( n \) means that a timeout occurred or the pipe went in reset state.

Function Class:

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

7.37.3.7 chPipeGetSize()

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

Returns the pipe buffer size as number of bytes.

Parameters

\[
\begin{array}{|c|c|}
\hline
\text{in} & pp & \text{the pointer to an initialized pipe_t object} \\
\hline
\end{array}
\]

Returns

The size of the pipe.

Function Class:

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

7.37.3.8 chPipeGetUsedCount()

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

Returns the number of used byte slots into a pipe.

Parameters

\[
\begin{array}{|c|c|}
\hline
\text{in} & pp & \text{the pointer to an initialized pipe_t object} \\
\hline
\end{array}
\]

Returns

The number of queued bytes.
Function Class:

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

7.37.3.9 chPipeGetFreeCount()  

\texttt{static size\_t chPipeGetFreeCount (  
\quad \text{\texttt{const pipe\_t \* pp}}} \text{[inline]}, \text{[static]}

Returns the number of free byte slots into a pipe.

Parameters

\begin{tabular}{|l|}
\hline
\textbf{in} \texttt{pp} \quad the pointer to an initialized pipe\_t object \\
\hline
\end{tabular}

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:

![Call Graph](call_graph.png)

7.37.3.10 chPipeResume()  

\texttt{static void chPipeResume (  
\quad \texttt{pipe\_t \* pp}}} \text{[inline]}, \text{[static]}

Terminates the reset state.
7.37 Pipes

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.
7.38 Delegate Threads

7.38.1 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.
- static msg_t chDelegateCallVeneer (thread_t *tp, delegate_veneer_t veneer,...)
  Triggers a function call on a delegate thread.
- void chDelegateDispatch (void)
  Call messages dispatching.
- msg_t chDelegateDispatchTimeout (sysinterval_t timeout)
  Call messages dispatching with timeout.
- static msg_t chDelegateCallDirect0 (thread_t *tp, delegate_fn0_t func)
  Direct call to a function with no parameters.
- static msg_t chDelegateCallDirect1 (thread_t *tp, delegate_fn1_t func, msg_t p1)
  Direct call to a function with one parameter.
- static msg_t chDelegateCallDirect2 (thread_t *tp, delegate_fn2_t func, msg_t p1, msg_t p2)
  Direct call to a function with two parameters.
- static msg_t chDelegateCallDirect3 (thread_t *tp, delegate_fn3_t func, msg_t p1, msg_t p2, msg_t p3)
  Direct call to a function with three parameters.
- static msg_t chDelegateCallDirect4 (thread_t *tp, delegate_fn4_t func, msg_t p1, msg_t p2, msg_t p3, msg_t p4)
  Direct call to a function with four parameters.
7.38 Delegate Threads

7.38.2 Typedef Documentation

7.38.2.1 delegate_veneer_t

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

7.38.2.2 delegate_fn0_t

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

7.38.2.3 delegate_fn1_t

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

7.38.2.4 delegate_fn2_t

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

7.38.2.5 delegate_fn3_t

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

7.38.2.6 delegate_fn4_t

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

7.38.3 Function Documentation

7.38.3.1 __ch_delegate_fn0()

msg_t __ch_delegate_fn0 {
    va_list * argsp
}
Veneer for functions with no parameters.
Parameters

in argsp the list of arguments

Returns

The function return value.

7.38.3.2 __ch_delegate_fn1()

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.

7.38.3.3 __ch_delegate_fn2()

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.

7.38.3.4 __ch_delegate_fn3()

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.

### 7.38.3.5 __ch_delegate_fn4()

```c
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.

### 7.38.3.6 chDelegateCallVeneer()

```c
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:

![Call Graph for chDelegateCallVeneer](image)

### 7.38.3.7 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:

![Call Graph for chDelegateDispatch](image)
7.38.3.8  chDelegateDispatchTimeout()

`msg_t chDelegateDispatchTimeout`

`sysinterval_t timeout`)

Call messages dispatching with timeout.

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

**Parameters**

<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>• <em>TIME_INFINITE</em> 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:
7.38.3.9 chDelegateCallDirect0()

static msg_t chDelegateCallDirect0 {
    thread_t * tp,
    delegate_fn0_t func } [inline], [static]

Direct call to a function with no parameters.

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

Parameters

| in | tp  | pointer to the delegate thread |
| in | func | pointer to the function to be called |

Returns
The function return value as a msg_t.

Here is the call graph for this function:

![](call_graph.png)

7.38.3.10 chDelegateCallDirect1()

static msg_t chDelegateCallDirect1 {
    thread_t * tp,
    delegate_fn1_t func,
    msg_t p1 } [inline], [static]

Direct call to a function with one parameter.

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

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>in</strong></td>
<td><em>tp</em></td>
<td>pointer to the delegate thread</td>
</tr>
<tr>
<td><strong>in</strong></td>
<td><em>func</em></td>
<td>pointer to the function to be called</td>
</tr>
<tr>
<td><strong>in</strong></td>
<td><em>p1</em></td>
<td>parameter 1 passed as a <em>msg_t</em></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
```

### 7.38.3.11 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

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

The function return value as a `msg_t`.

Here is the call graph for this function:

```
chDelegateCallDirect2
__ch_delegate_fn2
chDelegateCallVeneer chMsgSend
```

7.38.3.12 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.

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

7.38.3.13 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

<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>
<tr>
<td>in</td>
<td>p3</td>
<td>parameter 3 passed as a <code>msg_t</code></td>
</tr>
<tr>
<td>in</td>
<td>p4</td>
<td>parameter 4 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:
7.39 Jobs Queues

7.39.1 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.
7.39 Jobs Queues

7.39.2 Macro Definition Documentation

7.39.2.1 MSG_JOB_NULL

#define MSG_JOB_NULL ((msg_t)-2)

Dispatcher return code in case of a JOB_NULL has been received.

7.39.3 Typedef Documentation

7.39.3.1 jobs_queue_t

typedef struct ch_jobs_queue jobs_queue_t

Type of a jobs queue.

7.39.3.2 job_function_t

typedef void(* job_function_t) (void *arg)

Type of a job function.

7.39.3.3 job_descriptor_t

typedef struct ch_job_descriptor job_descriptor_t

Type of a job descriptor.

7.39.4 Function Documentation

7.39.4.1 chJobObjectInit()

static void chJobObjectInit ( 
    jobs_queue_t * jqp,
    size_t jobsn,
    job_descriptor_t * jobsbuf,
    msg_t * msgbuf ) [inline], [static]

Initializes a jobs queue object.
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>out</code> <code>jqp</code></td>
<td>pointer to a <code>jobs_queue_t</code> structure</td>
</tr>
<tr>
<td><code>in</code> <code>jobsn</code></td>
<td>number of jobs available</td>
</tr>
<tr>
<td><code>in</code> <code>jobsbuf</code></td>
<td>pointer to the buffer of jobs, it must be able to hold <code>jobsn job_descriptor_t</code> structures</td>
</tr>
<tr>
<td><code>in</code> <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:

```
chJobObjectInit
chGuardedPoolLoadArray
chGuardedPoolObjectInit
chMBObjectInit
chGuardedPoolAdd
chGuardedPoolObjectInitAligned
chThdQueueObjectInit
```

7.39.4.2 chJobGet()

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

Allocates a free job object.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>in</code> <code>jqp</code></td>
<td>pointer to a <code>jobs_queue_t</code> structure</td>
</tr>
</tbody>
</table>

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:

```
chJobGet -> chGuardedPoolAllocTimeout -> chGuardedPoolAllocTimeoutS
<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>chSysLock</td>
<td>chSysUnlock</td>
<td>chJobGet</td>
</tr>
</tbody>
</table>
```

### 7.39.4.3 chJobGet()

```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 Image]

### 7.39.4.4 chJobGetTimeoutS()

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

Allocates a free job object.

**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>• <strong>TIME_IMMEDIATE</strong> immediate timeout.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <strong>TIME_INFINITE</strong> no timeout.</td>
</tr>
</tbody>
</table>

**Returns**

The pointer to the allocated job object.

**Return values**

| **NULL** | if a job object is not available within the specified timeout. |
Function Class:

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

Here is the call graph for this function:

![Call Graph](chJobGetTimeoutS chGuardedPoolAllocTimeoutS chPoolAllocI chSemWaitTimeoutS)

### 7.39.4.5 chJobGetTimeout()

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

Allocates a free job object.

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</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>• <strong>TIME_IMMEDIATE</strong> immediate timeout.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <strong>TIME_INFINITE</strong> no timeout.</td>
</tr>
</tbody>
</table>

**Returns**

The pointer to the allocated job object.

**Return values**

<table>
<thead>
<tr>
<th>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:

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

Here is the call graph for this function:

```
chJobGetTimeout  --- chGuardedPoolAllocTimeout
              |               --- chSysLock
              |                |               --- chGuardedPoolAllocTimeoutS
chSysUnlock
```

7.39.4.6 chJobPostI()

```c
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

<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>jp</td>
<td>pointer to the job object to be posted</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:

```
chJobPostS
chMBPost
chDbgCheckClass
chMBGetFreeCount
chThdDequeueNext
```

### 7.39.4.7 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><code>in</code></td>
<td><code>jqp</code></td>
</tr>
<tr>
<td><code>in</code></td>
<td><code>jp</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.

Here is the call graph for this function:

![Call Graph for chJobPost()]

### 7.39.4.8 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>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>in</strong></td>
<td><strong>jqp</strong></td>
</tr>
<tr>
<td><strong>in</strong></td>
<td><strong>jp</strong></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]

7.39.4.9 chJobPostAheadI()

```c
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

<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>jp</td>
<td>pointer to the job object to be posted</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]

7.39.4.10 chJobPostAheadS()

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

Posts an high priority job object.

Note

By design the object can be always immediately posted.

Parameters

<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>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](image)

### 7.39.4.11 chJobPostAhead()

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

Posts an high priority job object.

**Note**

By design the object can be always immediately posted.

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td><code>jqp</code></td>
</tr>
<tr>
<td>in</td>
<td><code>jp</code></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]

7.39.4.12  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:

---

### 7.39.4.13 chJobDispatchTimeout()

```c
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

| MSG_OK         | if a job has been executed. |
| MSG_TIMEOUT    | if a timeout occurred. |
| 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:
7.40 Memory Management

7.40.1 Detailed Description

Memory Management services.

Modules

- Core Memory Manager
- Memory Heaps
- Memory Pools
7.41 Core Memory Manager

7.41.1 Detailed Description

Core Memory Manager related APIs and services.

Operation mode

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

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

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

By having a centralized memory provider the various allocators can coexist and share the main memory.

This allocator, alone, is also useful for very simple applications that just require a simple way to get memory blocks.

Precondition

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

Note

Compatible with RT and NIL.

Macros

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

Typedefs

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

Data Structures

- struct memcore_t
  Type of memory core object.
### Functions

- `void _core_init (void)`  
  _Low level memory manager initialization._

- `void * chCoreAllocFromBaseI (size_t size, unsigned align, size_t offset)`  
  _Allocates a memory block starting from the lowest address upward._

- `void * chCoreAllocFromTopI (size_t size, unsigned align, size_t offset)`  
  _Allocates a memory block starting from the top address downward._

- `void * chCoreAllocFromBase (size_t size, unsigned align, size_t offset)`  
  _Allocates a memory block starting from the lowest address upward._

- `void * chCoreAllocFromTop (size_t size, unsigned align, size_t offset)`  
  _Allocates a memory block starting from the top address downward._

- `size_t chCoreGetStatusX (void)`  
  _Core memory status._

- `static void * chCoreAllocAlignedI (size_t size, unsigned align)`  
  _Allocates a memory block._

- `static void * chCoreAllocAligned (size_t size, unsigned align)`  
  _Allocates a memory block._

- `static void * chCoreAllocI (size_t size)`  
  _Allocates a memory block._

- `static void * chCoreAlloc (size_t size)`  
  _Allocates a memory block._

### Variables

- `memcore_t ch_memcore`  
  _Memory core descriptor._

### 7.41.2 Macro Definition Documentation

#### 7.41.2.1 CH_CFG_MEMCORE_SIZE

```c
#define CH_CFG_MEMCORE_SIZE 0
```

_Manged 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`.

---

*ChibiOS/RT*
7.41.2.2 chCoreAllocAlignedWithOffsetI

#define chCoreAllocAlignedWithOffsetI chCoreAllocFromTopI

 Allocates a memory block.

Note

This is a generic form with unspecified allocation position.

Function Class:

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

7.41.2.3 chCoreAllocAlignedWithOffset

#define chCoreAllocAlignedWithOffset chCoreAllocFromTop

 Allocates a memory block.

Note

This is a generic form with unspecified allocation position.

Function Class:

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

7.41.3 Typedef Documentation

7.41.3.1 memgetfunc_t

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

Memory get function.

7.41.3.2 memgetfunc2_t

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

Enhanced memory get function.
7.41.4 Function Documentation

7.41.4.1 _core_init()

```c
void _core_init (  
        void  )
```

Low level memory manager initialization.

Function Class:

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

7.41.4.2 chCoreAllocFromBaseI()

```c
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 \( \text{offset} + \text{size} \) bytes. The returned pointer has \( \text{offset} \) bytes before its address and \( \text{size} \) 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:

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]

7.41.4.3 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>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:

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:

```
chCoreAllocFromTopI → chDbgCheckClassI → chSysHalt
```

7.41.4.4 chCoreAllocFromBase()

```c
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.
Function Class:
Normal API, 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]

7.41.4.5 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:

7.41.4.6 chCoreGetStatusX()

size_t chCoreGetStatusX (  
  void  )

Core memory status.

Returns

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

Function Class:

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

7.41.4.7 chCoreAllocAlignedI()

static void* chCoreAllocAlignedI (  
  size_t size,  
  unsigned align ) [inline], [static]

Allocates a memory block.

The allocated block is guaranteed to be properly aligned to the specified alignment.

Note

This is a generic form with unspecified allocation position.
Parameters

| in | 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:

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

7.41.4.8 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 |
7.41 Core Memory Manager

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

7.41.4.9 chCoreAllocI()

static void* chCoreAllocI (size_t size) [inline], [static]

Allocates a memory block.
The allocated block is guaranteed to be properly aligned for a pointer data type.

Note
This is a generic form with unspecified allocation position.

Parameters

| in  | size   | the size of the block to be allocated. |

Returns
A pointer to the allocated memory block.

Return values

| NULL | allocation failed, core memory exhausted. |

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

7.41.4.10 chCoreAlloc()

static void* chCoreAlloc (size_t size) [inline], [static]

Allocates a memory block.
The allocated block is guaranteed to be properly aligned for a pointer data type.

Note
This is a generic form with unspecified allocation position.
Parameters

| in  | size | the size of the block to be allocated |

Returns

A pointer to the allocated memory block.

Return values

| NULL | allocation failed, core memory exhausted |

Function Class:

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

7.41.5 Variable Documentation

7.41.5.1 ch_memcore

memcore_t ch_memcore

Memory core descriptor.
7.42 Memory Heaps

7.42.1 Detailed Description

Heap Allocator related APIs.

Operation mode

The heap allocator implements a first-fit strategy and its APIs are functionally equivalent to the usual malloc() and free() library functions. The main difference is that the OS heap APIs are guaranteed to be thread safe and there is the ability to return memory blocks aligned to arbitrary powers of two.

Precondition

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

Note

Compatible with RT and NIL.

Macros

- #define CH_HEAP_ALIGNMENT 8U
  
  Minimum alignment used for heap.
- #define CH_HEAP_AREA(name, size)

  Allocation of an aligned static heap buffer.

Typedefs

- typedef struct memory_heap memory_heap_t
  
  Type of a memory heap.
- typedef union heap_header heap_header_t

  Type of a memory heap header.

Data Structures

- union heap_header

  Memory heap block header.
- struct memory_heap

  Structure describing a memory heap.
Functions

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

Variables

- **static memory_heap_t default_heap**
  
  Default heap descriptor.

7.42.2 Macro Definition Documentation

7.42.2.1 CH_HEAP_ALIGNMENT

```c
#define CH_HEAP_ALIGNMENT 8U
```

Minimum alignment used for heap.

Note

Cannot use the sizeof operator in this macro.

7.42.2.2 CH_HEAP_AREA

```c
#define CH_HEAP_AREA(
  name,
  size
)
```

Value:

```c
ALIGNED_VAR(CH_HEAP_ALIGNMENT)
uint8_t name[MEM_ALIGN_NEXT((size), CH_HEAP_ALIGNMENT)]
```

Allocation of an aligned static heap buffer.
7.42 Memory Heaps

7.42.3 Typedef Documentation

7.42.3.1 memory_heap_t

typedef struct memory_heap memory_heap_t

Type of a memory heap.

7.42.3.2 heap_header_t

typedef union heap_header heap_header_t

Type of a memory heap header.

7.42.4 Function Documentation

7.42.4.1 _heap_init()

void _heap_init ( void )

Initializes the default heap.

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

7.42.4.2 chHeapObjectInit()

void chHeapObjectInit ( memory_heap_t * heapp,
                        void * buf,
                        size_t size )

Initializes a memory heap from a static memory area.

Note
The heap buffer base and size are adjusted if the passed buffer is not aligned to CH_HEAP_ALIGNMENT. This mean that the effective heap size can be less than size.
Parameters

<table>
<thead>
<tr>
<th>out</th>
<th>heap</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.

7.42.4.3 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>heap</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.

7.42.4.4 chHeapFree()

```c
void chHeapFree ( 
    void * p )
```

Frees a previously allocated memory block.
Parameters

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>in</strong></td>
<td><strong>heapp</strong></td>
<td>pointer to a heap descriptor or <strong>NULL</strong> in order to access the default heap.</td>
<td></td>
</tr>
<tr>
<td><strong>in</strong></td>
<td><strong>totalp</strong></td>
<td>pointer to a variable that will receive the total fragmented free space or <strong>NULL</strong></td>
<td></td>
</tr>
<tr>
<td><strong>in</strong></td>
<td><strong>largestp</strong></td>
<td>pointer to a variable that will receive the largest free block found space or <strong>NULL</strong></td>
<td></td>
</tr>
</tbody>
</table>

Returns

The number of fragments in the heap.

Function Class:

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

7.42.4.6 chHeapAlloc()

```
static void* chHeapAlloc (  
    memory_heap_t * heapp,  
    size_t size ) [inline], [static]
```

Allocates a block of memory from the heap by using the first-fit algorithm.

The allocated block is guaranteed to be properly aligned for a pointer data type.
Parameters

| in | 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:

7.42.4.7 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.

7.42.5 Variable Documentation

7.42.5.1 default_heap

`memory_heap_t default_heap [static]`

Default heap descriptor.
7.43 Memory Pools

7.43.1 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 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 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 chPoolAddS (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 * chGuardedPoolAlloc (guarded_memory_pool_t *gmp)
  Allocates an object from a guarded memory pool.

• static void chGuardedPoolFree (guarded_memory_pool_t *gmp, void *objp)
  Releases an object into a guarded memory pool.

• static void chGuardedPoolFreeI (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.

7.43.2 Macro Definition Documentation
7.43.2.1  _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>name</td>
<td>the name of the memory pool variable</td>
</tr>
<tr>
<td>size</td>
<td>size of the memory pool contained objects</td>
</tr>
<tr>
<td>align</td>
<td>required memory alignment</td>
</tr>
<tr>
<td>provider</td>
<td>memory provider function for the memory pool</td>
</tr>
</tbody>
</table>

7.43.2.2  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>name</td>
<td>the name of the memory pool variable</td>
</tr>
<tr>
<td>size</td>
<td>size of the memory pool contained objects</td>
</tr>
<tr>
<td>align</td>
<td>required memory alignment</td>
</tr>
<tr>
<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>

7.43.2.3  _GUARDEDMEMORYPOOL_DATA

#define _GUARDEDMEMORYPOOL_DATA(
    name,
    size,
    align )
Value:
{
    _SEMAPHORE_DATA(name.sem, (cnt_t)0),
    _MEMORYPOOL_DATA(NULL, size, align, NULL)
}

Data part of a static guarded memory pool initializer.

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

Parameters

<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>

### 7.43.2.4 GUARDEDMEMORYPOOL_DECL

#define GUARDEDMEMORYPOOL_DECL(
    name,
    size,
    align
) guarded_memory_pool_t name = _GUARDEDMEMORYPOOL_DATA(name, size, align)

Static guarded memory pool initializer.

Statically initialized guarded memory pools require no explicit initialization using chGuardedPoolInit().

Parameters

<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>

### 7.43.3 Function Documentation

#### 7.43.3.1 chPoolObjectInitAligned()

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

Initializes an empty memory pool.
Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td><em>mp</em></td>
<td>pointer to a <code>memory_pool_t</code> structure</td>
</tr>
<tr>
<td>in</td>
<td><em>size</em></td>
<td>the size of the objects contained in this memory pool, the minimum accepted size is the size of a pointer to <code>void</code>.</td>
</tr>
<tr>
<td>in</td>
<td><em>align</em></td>
<td>required memory alignment</td>
</tr>
<tr>
<td>in</td>
<td><em>provider</em></td>
<td>memory provider function for the memory pool or <code>NULL</code> 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.

7.43.3.2 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>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td><em>mp</em></td>
<td>pointer to a <code>memory_pool_t</code> structure</td>
</tr>
<tr>
<td>in</td>
<td><em>p</em></td>
<td>pointer to the array first element</td>
</tr>
<tr>
<td>in</td>
<td><em>n</em></td>
<td>number of elements in the array</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)

### 7.43.3.3 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:

```
chPoolAllocI chDbgCheckClassI chSysHalt
```

### 7.43.3.4 chPoolAlloc()

```c
void * chPoolAlloc (  
    memory_pool_t * mp )
```

Allocates an object from a memory pool.

**Precondition**

The memory pool must already be initialized.

**Parameters**

- `mp` (memory_pool_t *) pointer to a memory_pool_t structure

**Returns**

- The pointer to the allocated object.

**Return values**

- `NULL` if pool is empty.
7.43 Memory Pools

Function Class:

Normal API, 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)

7.43.3.5 chPoolFreeI()

```c
void chPoolFreeI ( memory_pool_t * mp,
                   void * objp )
```

Releases an object into a memory pool.

Precondition

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

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in mp</td>
<td>pointer to a <code>memory_pool_t</code> structure</td>
</tr>
<tr>
<td>in objp</td>
<td>the pointer to the object to be released</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:

```
chPoolFreeI  chDbgCheckClassI  chSysHalt
```

7.43.3.6 chPoolFree()

```c
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:

```
7.43.3.7 chGuardedPoolObjectInitAligned()

void chGuardedPoolObjectInitAligned (  
guarded_memory_pool_t * gmp,  
size_t size,  
unsigned align )

Initializes an empty guarded memory pool.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>out gmp</td>
<td>pointer to a guarded_memory_pool_t structure</td>
</tr>
<tr>
<td>in 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>
<tr>
<td>in align</td>
<td>required memory alignment</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](image)

7.43.3.8 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>gmp</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:

![Call Graph]

7.43.3.9 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><code>in gmp</code></td>
<td>pointer to a <code>guarded_memory_pool_t</code> structure</td>
</tr>
<tr>
<td><code>in timeout</code></td>
<td>the number of ticks before the operation times out, the following special values are allowed:</td>
</tr>
<tr>
<td></td>
<td></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

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>NULL</code></td>
<td>if the operation 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:

```
chGuardedPoolAllocTimeoutS
  ┌────────┐
  │        │
  │ chPoolAlloc │
  │           │
  │           │
  │ chDbgCheckClassI │
  │           │
  │           │
  │ ch_queue_isempty │
  │           │
  │           │
  │ chSemWaitTimeoutS │
  │           │
  │           │
  │ ch_queue_notempty │
  │           │
  │           │
  │ chDbgCheckClassS │
```

7.43.3.10  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>in</th>
<th>gmp</th>
<th>pointer to a <code>guarded_memory_pool_t</code> 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>• <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 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.

Here is the call graph for this function:

![Call Graph Image]

### 7.43.3.11 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>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td><code>gmp</code> pointer to a <code>guarded_memory_pool_t</code> structure</td>
</tr>
<tr>
<td>in</td>
<td><code>objp</code> 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:

```
chGuardedPoolFree
chGuardedPoolFreeI
chSchRescheduleS
chSysLock
chSysUnlock
chPoolFreeI
chSemSignalI
chDbgCheckClassS
chSchDoRescheduleAhead
_dbg_check_lock
_stats_start_measure
_crit_thd
_dbg_check_unlock
_stats_stop_measure
_crit_thd
```

### 7.43.3.12 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 <code>memory_pool_t</code> 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:

```
chPoolObjectInit -> chPoolObjectInitAligned
```

### 7.43.3.13 chPoolAdd()

```c
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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>mp</code></td>
<td>pointer to a <code>memory_pool_t</code> structure</td>
</tr>
<tr>
<td><code>objp</code></td>
<td>the pointer to the object to be added</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]

7.43.3.14 chPoolAddI()

```c
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>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.
7.43 Memory Pools

Here is the call graph for this function:

![Call Graph]

### 7.43.3.15 chGuardedPoolObjectInit()

```c
static void chGuardedPoolObjectInit ( safeguarded_memory_pool_t * gmp,
                        size_t size ) [inline], [static]
```

Initializes an empty guarded memory pool.

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>gmp</td>
</tr>
<tr>
<td>in</td>
<td>size</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]

### 7.43.3.16 chGuardedPoolGetCounterI()

```c
static cnt_t chGuardedPoolGetCounterI ( safeguarded_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.

Here is the call graph for this function:

7.43.3.17  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:

![Call Graph Image]

---

**7.43.3.18 chGuardedPoolFreeI()**

```c
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

<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>

Function 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]

7.43.3.19  chGuardedPoolFreeS()

```c
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>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:

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

Here is the call graph for this function:

7.43.3.20  **chGuardedPoolAdd()**

```c
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><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 added</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]

### 7.43.3.21 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>Mode</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](image)

### 7.43.3.22 chGuardedPoolAddS()

```c
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**
<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>in</code></td>
<td><code>gmp</code></td>
</tr>
<tr>
<td><code>in</code></td>
<td><code>objp</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.

Here is the call graph for this function:
7.44 Complex Services

7.44.1 Detailed Description

Modules

- Objects FIFOs
- Objects Caches
- Dynamic Objects Factory
7.45 Objects FIFOs

7.45.1 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 a high priority object.

- static void chFifoSendObjectAheadS (objects_fifo_t *ofp, void *objp)
  Posts a high priority object.

- static void chFifoSendObjectAhead (objects_fifo_t *ofp, void *objp)
  Posts a high priority object.

- static msg_t chFifoReceiveObjectI (objects_fifo_t *ofp, void **objpp)
  Fetches an object.

- static msg_t chFifoReceiveObjectTimeoutS (objects_fifo_t *ofp, void **objpp, sysinterval_t timeout)
  Fetches an object.

- static msg_t chFifoReceiveObjectTimeout (objects_fifo_t *ofp, void **objpp, sysinterval_t timeout)
  Fetches an object.
7.45 Objects FIFOs

7.45.2 Typedef Documentation

7.45.2.1 objects_fifo_t

typedef struct ch_objects_fifo objects_fifo_t

Type of an objects FIFO.

7.45.3 Function Documentation

7.45.3.1 chFifoObjectInitAligned()

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

Initializes a FIFO object.

Precondition

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

Parameters

<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](image)

7.45.3.2 chFifoObjectInit()

```c
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>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ofp</code></td>
<td>pointer to a <code>objects_fifo_t</code> structure</td>
</tr>
<tr>
<td><code>objsize</code></td>
<td>size of objects</td>
</tr>
<tr>
<td><code>objn</code></td>
<td>number of objects available</td>
</tr>
<tr>
<td><code>objbuf</code></td>
<td>pointer to the buffer of objects, it must be able to hold <code>objn</code> objects of <code>objsize</code> size</td>
</tr>
<tr>
<td><code>msgbuf</code></td>
<td>pointer to the buffer of messages, it must be able to hold <code>objn</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:

7.45.3.3 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  chSemFastWaitI  chSemGetCounterI
```

7.45.3.4 **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 <em>objects_fifo_t</em> 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>• <strong>TIME_IMMEDIATE</strong> immediate timeout.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <strong>TIME_INFINITE</strong> 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.
7.45 Objects FIFOs

Here is the call graph for this function:

```
chFifoTakeObjectTimeoutS
<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
</table>
| chGuardedPoolAllocTimeoutS
<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
</table>
| chPoolAllocS
<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
</table>
| chSemWaitTimeoutS
```

7.45.3.5 chFifoTakeObjectTimeout()

```
static void∗ chFifoTakeObjectTimeout (  
    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:

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

Here is the call graph for this function:

```
<table>
<thead>
<tr>
<th>chFifoTakeObjectTimeout</th>
<th>chGuardedPoolAllocTimeout</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>chSysLock</td>
</tr>
<tr>
<td></td>
<td>chSysUnlock</td>
</tr>
</tbody>
</table>
```

7.45.3.6 chFifoReturnObjectI()

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

Releases a fetched object.

<table>
<thead>
<tr>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>in  ofp</td>
</tr>
<tr>
<td>in  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.

Here is the call graph for this function:

```
<table>
<thead>
<tr>
<th>chFifoReturnObjectI</th>
<th>chGuardedPoolFree</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>chSemSignall</td>
</tr>
<tr>
<td></td>
<td>chPoolFreeel</td>
</tr>
</tbody>
</table>
```
7.45 Objects FIFOs

7.45.3.7  chFifoReturnObjectS()

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

Releases a fetched 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>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:

```
chFifoReturnObjectS  
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

7.45.3.8  chFifoReturnObject()

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

Releases a fetched 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>objp</td>
<td>pointer to the object to be released</td>
</tr>
</tbody>
</table>

ChibiOS/RT
Function Class:

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

Here is the call graph for this function:

7.45.3.9 chFifoSendObjectI()

static void chFifoSendObjectI (
    object_fifo_t * ofp,
    void * objp ) [inline], [static]

Posts an 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</td>
<td>ofp</td>
</tr>
<tr>
<td>in</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.

Here is the call graph for this function:

```
chFifoSendObjectI  chMBPostI  chDbgCheckClassI
                   chMBGetFreeCountI
                chThdDequeueNextI
```

### 7.45.3.10 chFifoSendObjectS()

```c
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**

<table>
<thead>
<tr>
<th></th>
<th><strong>ofp</strong></th>
<th>pointer to a objects_fifo_t 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>

ChibiOS/RT
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)

7.45.3.11  **chFifoSendObject()**

```c
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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>in ofp</code></td>
<td>pointer to a <code>objects_fifo_t</code> structure</td>
</tr>
<tr>
<td><code>in objp</code></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:

7.45.3.12 chFifoSendObjectAheadI()

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

Posts an high priority object.

Note

By design the object can be always immediately posted.

Parameters

<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 posted</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)

### 7.45.3.13 chFifoSendObjectAheadS()

```c
static void chFifoSendObjectAheadS(
    objects_fifo_t *ofp,
    void *objp)
```

Posts an high priority 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><code>ofp</code></td>
<td>pointer to a <code>objects_fifo_t</code> structure</td>
</tr>
<tr>
<td><code>objp</code></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:

![Call Graph](image)
7.45.3.14 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>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>ofp</td>
<td>pointer to a objects_fifo_t structure</td>
</tr>
<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](call_graph.png)

7.45.3.15 chFifoReceiveObjectI()

static msg_t chFifoReceiveObjectI ( 
    objects_fifo_t * ofp, 
    void ** objpp ) [inline], [static]

Fetches 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 objects_fifo_t structure</td>
</tr>
<tr>
<td>in</td>
<td>objpp</td>
<td>pointer to the fetched object reference</td>
</tr>
</tbody>
</table>
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 FIFO 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:

```
chFifoReceiveObjectI -> chMBFetchI -> chMBGetUsedCountI -> chDbgCheckClassI
```

7.45.3.16 chFifoReceiveObjectTimeoutS()

```c
static msg_t chFifoReceiveObjectTimeoutS ( 
objects_fifo_t * ofp,
void ** objpp,
syinterval_t timeout ) [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>
<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_INFINITY no timeout.</td>
</tr>
</tbody>
</table>

Returns
The operation status.
Return values

| MSG_OK       | if an object has been correctly fetched. |
| 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 Image]

### 7.45.3.17 chFifoReceiveObjectTimeout()

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

Fetches an object.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th><strong>ofp</strong></th>
<th>pointer to a <strong>objects_fifo_t</strong> structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td><strong>objpp</strong></td>
<td>pointer to the fetched object reference</td>
</tr>
<tr>
<td>in</td>
<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></td>
<td>• <strong>TIME_IMMEDIATE</strong> immediate timeout.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <strong>TIME_INFINITE</strong> no timeout.</td>
</tr>
</tbody>
</table>

**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:
7.46 Objects Caches

7.46.1 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_t)(objects_cache_t ∗ocp, oc_object_t ∗objp, bool async)
  Object read function.
- typedef bool(∗oc_writef_t)(objects_cache_t ∗ocp, oc_object_t ∗objp, bool async)
  Object write function.

Data Structures

- struct ch_oc_hash_header
  Structure representing a hash table element.
- struct ch_oc_lru_header
  Structure representing an hash table element.
- struct ch_oc_object
  Structure representing a cached object.
- struct ch_objects_cache
  Structure representing a cache object.
Functions

- static oc_object_t * hash_get_s (objects_cache_t *ocp, uint32_t group, uint32_t key)
  
  Returns an object pointer from the cache, if present.

- static oc_object_t * lru_get_last_s (objects_cache_t *ocp)
  
  Gets the least recently used object buffer from the LRU list.

- void chCacheObjectInit (objects_cache_t *ocp, ucnt_t hashn, oc_hash_header_t *hashp, ucnt_t objn, size_t objsz, void *objvp, oc_readf_t readf, oc_writef_t writef)
  
  Initializes a objects_cache_t object.

- oc_object_t * chCacheGetObject (objects_cache_t *ocp, uint32_t group, uint32_t key)
  
  Retrieves an object from the cache.

- void chCacheReleaseObject (objects_cache_t *ocp, oc_object_t *objp)
  
  Releases an object into the cache.

- bool chCacheReadObject (objects_cache_t *ocp, oc_object_t *objp, bool async)
  
  Reads object data from the storage.

- bool chCacheWriteObject (objects_cache_t *ocp, oc_object_t *objp, bool async)
  
  Writes the object data back to storage.

- static void chCacheReleaseObject (objects_cache_t *ocp, oc_object_t *objp)
  
  Releases an object into the cache.

7.46.2 Typedef Documentation

7.46.2.1 oc_flags_t

typedef uint32_t oc_flags_t

Flags of cached objects.

7.46.2.2 oc_hash_header_t

typedef struct ch_oc_hash_header oc_hash_header_t

Type of an hash element header.

7.46.2.3 oc_lru_header_t

typedef struct ch_oc_lru_header oc_lru_header_t

Type of an LRU element header.
7.46.2.4 oc_object_t

typedef struct ch_oc_object oc_object_t

Type of a cached object.

7.46.2.5 objects_cache_t

typedef struct ch_objects_cache objects_cache_t

Type of a cache object.

7.46.2.6 oc_readf_t

typedef bool(* oc_readf_t) (objects_cache_t *ocp, oc_object_t *objp, bool async)

Object read function.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>ocp</th>
<th>pointer to the objects_cache_t structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>async</td>
<td>requests an asynchronous operation if supported, the function is then responsible for releasing the object</td>
</tr>
</tbody>
</table>

7.46.2.7 oc_writef_t

typedef bool(* oc_writef_t) (objects_cache_t *ocp, oc_object_t *objp, bool async)

Object write function.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>ocp</th>
<th>pointer to the objects_cache_t structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>async</td>
<td>requests an asynchronous operation if supported, the function is then responsible for releasing the object</td>
</tr>
</tbody>
</table>

7.46.3 Function Documentation

ChibiOS/RT
### 7.46.3.1 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**

- **out ocp**: pointer to the `objects_cache_t` structure to be
- **in group**: object group identifier
- **in key**: object identifier within the group initialized

**Return values**

- **NULL**: if the object is not in cache.

**Function Class:**

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

### 7.46.3.2 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**

- **out ocp**: pointer to the `objects_cache_t` structure to be

**Returns**

- The pointer to the retrieved object.

**Function Class:**

Not an API, this function is for internal use only.
7.46.3.3 **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>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>out</strong> ocp</td>
<td>pointer to the <code>objects_cache_t</code> structure to be initialized</td>
</tr>
<tr>
<td>in hashn</td>
<td>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 hashp</td>
<td>pointer to the hash table as an array of <code>oc_hash_header_t</code></td>
</tr>
<tr>
<td>in objn</td>
<td>number of elements in the objects table array</td>
</tr>
<tr>
<td>in objsz</td>
<td>size of elements in the objects table array, the minimum value is <code>sizeof (oc_object_t)</code></td>
</tr>
<tr>
<td>in objvp</td>
<td>pointer to the hash objects as an array of structures starting with an <code>oc_object_t</code></td>
</tr>
<tr>
<td>in readf</td>
<td>pointer to an object reader function</td>
</tr>
<tr>
<td>in writef</td>
<td>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.

Here is the call graph for this function:

![Call Graph](image)

7.46.3.4 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>in</th>
<th>ocp</th>
<th>pointer to the objects_cache_t structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>group</td>
<td>object group identifier</td>
</tr>
<tr>
<td>in</td>
<td>key</td>
<td>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:

```
    chCacheGetObject
    chSysLock
    hash_get_s
    _dbg_check_lock
    _stats_start_measure
    _crit_thd
```

7.46.3.5 chCacheReleaseObjectI()

```c
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

<table>
<thead>
<tr>
<th>in</th>
<th>ocp</th>
<th>pointer to the objects_cache_t structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>objp</td>
<td>pointer to the oc_object_t structure</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.
7.46.3.6  chCacheReadObject()

bool chCacheReadObject (  
    objects_cache_t  * ocp,  
    oc_object_t  * objp,  
    bool async )  

Reads object data from the storage.

Note  
In case of asynchronous operation an error condition is not reported by this function.

Parameters  

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>ocp</td>
<td>pointer to the objects_cache_t structure</td>
</tr>
<tr>
<td>in</td>
<td>obj</td>
<td>pointer to the oc_object_t structure</td>
</tr>
<tr>
<td>in</td>
<td>async</td>
<td>requests an asynchronous operation if supported, the function is then responsible for releasing the object</td>
</tr>
</tbody>
</table>

Returns  
The operation status. In case of asynchronous operation false is always returned.

Return values  

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></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.

7.46.3.7  chCacheWriteObject()

bool chCacheWriteObject (  
    objects_cache_t  * ocp,  
    oc_object_t  * objp,  
    bool async )  

Writes the object data back to storage.

Note  
In case of asynchronous operation an error condition is not reported by this function.
7.46 Objects Caches

Parameters

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>in</strong></td>
<td><strong>ocp</strong></td>
</tr>
<tr>
<td><strong>in</strong></td>
<td><strong>objp</strong></td>
</tr>
<tr>
<td><strong>in</strong></td>
<td><strong>async</strong></td>
</tr>
</tbody>
</table>

Returns

The operation status. In case of asynchronous operation false is always returned.

Return values

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>false</strong></td>
<td>if the operation succeeded.</td>
</tr>
<tr>
<td><strong>true</strong></td>
<td>if the synchronous write 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.

7.46.3.8 chCacheReleaseObject()

static void chCacheReleaseObject ( 
    objects_cache_t * ocp, 
    oc_object_t * objp ) [inline], [static]

Releases an object into the cache.

Note

This function gives a meaning to the following flags:

- OC_FLAG_INLRU must be cleared.
- OC_FLAG_INHASH must be set.
- OC_FLAG_SHARED must be cleared.
- OC_FLAG_NOTSYNC invalidates the object and queues it on the LRU tail.
- OC_FLAG_LAZYWRITE is ignored and kept, a write will occur when the object is removed from the LRU list (lazy write).

Parameters

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>in</strong></td>
<td><strong>ocp</strong></td>
</tr>
<tr>
<td><strong>in</strong></td>
<td><strong>objp</strong></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:
7.47 Dynamic Objects Factory

7.47.1 Detailed Description

The object factory is a subsystem that allows to:

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

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

Precondition

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

Note

Compatible with RT and NIL.

Macros

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

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

**Data Structures**

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

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

Variables

• objects_factory_t ch_factory
  Factory object static instance.

7.47.2  Macro Definition Documentation

7.47.2.1  CH_CFG_FACTORY_MAX_NAMES_LENGTH

#define CH_CFG_FACTORY_MAX_NAMES_LENGTH 8
Maximum length for object names.
If the specified length is zero then the name is stored by pointer but this could have unintended side effects.

7.47.2.2  CH_CFG_FACTORY_OBJECTS_REGISTRY

#define CH_CFG_FACTORY_OBJECTS_REGISTRY TRUE
Enables the registry of generic objects.

7.47.2.3  CH_CFG_FACTORY GENERIC BUFFERS

#define CH_CFG_FACTORY GENERIC BUFFERS TRUE
Enables factory for generic buffers.

7.47.2.4  CH_CFG_FACTORY_SEMAPHORES [1/2]

#define CH_CFG_FACTORY_SEMAPHORES TRUE
Enables factory for semaphores.
7.47 Dynamic Objects Factory

7.47.2.5 CH_CFG_FACTORY_SEMAPHORES [2/2]

#define CH_CFG_FACTORY_SEMAPHORES FALSE

Enables factory for semaphores.

7.47.2.6 CH_CFG_FACTORY_MAILBOXES [1/2]

#define CH_CFG_FACTORY_MAILBOXES TRUE

Enables factory for mailboxes.

7.47.2.7 CH_CFG_FACTORY_MAILBOXES [2/2]

#define CH_CFG_FACTORY_MAILBOXES FALSE

Enables factory for mailboxes.

7.47.2.8 CH_CFG_FACTORY_OBJ_FIFOS [1/3]

#define CH_CFG_FACTORY_OBJ_FIFOS TRUE

Enables factory for objects FIFOs.

7.47.2.9 CH_CFG_FACTORY_OBJ_FIFOS [2/3]

#define CH_CFG_FACTORY_OBJ_FIFOS TRUE

Enables factory for objects FIFOs.

7.47.2.10 CH_CFG_FACTORY_OBJ_FIFOS [3/3]

#define CH_CFG_FACTORY_OBJ_FIFOS FALSE

Enables factory for objects FIFOs.
7.47.2.11  CH_CFG_FACTORY_PIPES [1/2]

#define CH_CFG_FACTORY_PIPES TRUE

Enables factory for Pipes.

7.47.2.12  CH_CFG_FACTORY_PIPES [2/2]

#define CH_CFG_FACTORY_PIPES FALSE

Enables factory for Pipes.

7.47.3  Typedef Documentation

7.47.3.1  dyn_element_t

typedef struct ch_dyn_element dyn_element_t

Type of a dynamic object list element.

7.47.3.2  dyn_list_t

typedef struct ch_dyn_list dyn_list_t

Type of a dynamic object list.

7.47.3.3  registered_object_t

typedef struct ch_registered_static_object registered_object_t

Type of a registered object.

7.47.3.4  dyn_buffer_t

typedef struct ch_dyn_object dyn_buffer_t

Type of a dynamic buffer object.
7.47 Dynamic Objects Factory

7.47.3.5 dyn_semaphore_t

typedef struct ch_dyn_semaphore dyn_semaphore_t

Type of a dynamic semaphore.

7.47.3.6 dyn_mailbox_t

typedef struct ch_dyn_mailbox dyn_mailbox_t

Type of a dynamic buffer object.

7.47.3.7 dyn_objects_fifo_t

typedef struct ch_dyn_objects_fifo dyn_objects_fifo_t

Type of a dynamic buffer object.

7.47.3.8 dyn_pipe_t

typedef struct ch_dyn_pipe dyn_pipe_t

Type of a dynamic pipe object.

7.47.3.9 objects_factory_t

typedef struct ch_objects_factory objects_factory_t

Type of the factory main object.

7.47.4 Function Documentation
7.47.4.1 _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.

Here is the call graph for this function:

![Call graph](call_graph.png)

7.47.4.2 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>Name</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.

### 7.47.4.3 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.

### 7.47.4.4 chFactoryFindObjectByPointer()

```c
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   | objp | pointer to the object to be retrieved |

Returns

The reference to the found registered object.

Return values

| NULL | if a registered object with the specified pointer does not exist. |

Function Class:

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

7.4.7.4.5 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

| in   | rop | registered object reference |

Function Class:

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

7.4.7.4.6 chFactoryCreateBuffer()

```c
dyn_buffer_t * chFactoryCreateBuffer ( 
    const char * name,
    size_t size )
```

Creates a generic dynamic buffer object.
7.47 Dynamic Objects Factory

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.

7.47.4.7 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.

### 7.47.4.8 chFactoryReleaseBuffer()

```c
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.

### 7.47.4.9 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.

7.47.4.10 chFactoryFindSemaphore()

dyn_semaphore_t * chFactoryFindSemaphore ( const char * name )

Retrieves a dynamic semaphore object.

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

Parameters

| 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.

7.47.4.11 chFactoryReleaseSemaphore()

void chFactoryReleaseSemaphore ( dyn_semaphore_t * dsp )

Releases a dynamic semaphore object.
The reference counter of the dynamic semaphore object is decreased by one, if reaches zero then the dynamic semaphore object memory is freed.
Parameters

| in | 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.

### 7.47.4.12 chFactoryCreateMailbox()

```
dyn_mailbox_t * chFactoryCreateMailbox (const char * name, size_t n)
```

Creates a dynamic mailbox object.

**Postcondition**

A reference to the dynamic mailbox object is returned and the reference counter is initialized to one.

The dynamic mailbox object is initialized and ready to use.

**Parameters**

| in | 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.

### 7.47.4.13 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. |

Function Class:

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

7.47.4.14 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.

7.47.4.15 chFactoryCreateObjectsFIFO()

```c
dyn_objects_fifo_t * chFactoryCreateObjectsFIFO ( 
    const char * name,
    size_t objsize,
    size_t objn,
    unsigned objalign )
```

Creates a dynamic "objects FIFO" object.

ChibiOS/RT
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.

7.47.4.16 chFactoryFindObjectFIFO()

dyn_objects_fifo_t * chFactoryFindObjectFIFO ( 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.

7.47.4.17 chFactoryReleaseObjectsFIFO()

```c
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.

7.47.4.18 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.

7.47.4.19 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.

7.47.4.20 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

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>dep</td>
<td>dynamic pipe object reference</td>
</tr>
</tbody>
</table>

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

### 7.47.4.21 chFactoryDuplicateReference()

```c
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**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>dep</td>
<td>pointer to the element field of the object</td>
</tr>
</tbody>
</table>

**Returns**
The duplicated object reference.

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

### 7.47.4.22 chFactoryGetObject()

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

Returns the pointer to the inner registered object.

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>rop</td>
<td>registered object reference</td>
</tr>
</tbody>
</table>
7.47 Dynamic Objects Factory

Returns

The pointer to the registered object.

Function Class:

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

### 7.47.4.23 chFactoryGetBufferSize()

```c
static size_t chFactoryGetBufferSize (dyn_buffer_t * dbp) [inline], [static]
```

Returns the size of a generic dynamic buffer object.

**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:

![Call Graph]

### 7.47.4.24 chFactoryGetBuffer()

```c
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.

### 7.47.4.25 chFactoryGetSemaphore()

```c
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.

### 7.47.4.26 chFactoryGetMailbox()

```c
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.

7.47.4.27 chFactoryGetObjectsFIFO()

static objects_fifo_t* chFactoryGetObjectsFIFO (  
    dyn_objects_fifo_t * dof ) [inline], [static]

Returns the pointer to the inner objects FIFO.

Parameters

| in  | dof | dynamic "objects FIFO" object reference |

Returns

The pointer to the objects FIFO.

Function Class:

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

7.47.4.28 chFactoryGetPipe()

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

Returns the pointer to the inner pipe.

Parameters

| in  | dpp | dynamic pipe object reference |

Returns

The pointer to the pipe.

Function Class:

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

7.47.5.1 ch_factory

```c
objects_factory_t ch_factory
```

Factory object static instance.

Note

It is a global object because it could be accessed through a specific debugger plugin.
Chapter 8

Data Structure Documentation

8.1 ch_binary_semaphore Struct Reference

Binary semaphore type.

#include <chbsem.h>

Inheritance diagram for ch_binary_semaphore:
Collaboration diagram for `ch_binary_semaphore`:

```
+-----+          +-----+          +-----+
|      |          |      |          |      |
|      |          |      |          |      |
|      |          |      |          |      |
|      |          |      |          |      |
|      |          |      |          |      |
|      |          |      |          |      |
|      |          |      |          |      |
|      |          |      |          |      |
|      |          |      |          |      |
|      |          |      |          |      |
|      |          |      |          |      |
```

**Additional Inherited Members**

**8.1.1 Detailed Description**

Binary semaphore type.

**8.2 `ch_delta_list` Struct Reference**

Virtual Timer delta list element and header structure.

```
#include <chschd.h>
```

Collaboration diagram for `ch_delta_list`:

```
+-----+          +-----+          +-----+
|      |          |      |          |      |
|      |          |      |          |      |
|      |          |      |          |      |
|      |          |      |          |      |
|      |          |      |          |      |
|      |          |      |          |      |
|      |          |      |          |      |
|      |          |      |          |      |
|      |          |      |          |      |
|      |          |      |          |      |
```

ChibiOS/RT
8.2 ch_delta_list Struct Reference

Data Fields

- `delta_list_t * next`
  Next timer in the list.

- `delta_list_t * prev`
  Previous timer in the list.

- `sysinterval_t delta`
  Time delta before timeout.

8.2.1 Detailed Description

Virtual Timer delta list element and header structure.

8.2.2 Field Documentation

8.2.2.1 next

`delta_list_t* ch_delta_list::next`

Next timer in the list.

8.2.2.2 prev

`delta_list_t* ch_delta_list::prev`

Previous timer in the list.

8.2.2.3 delta

`sysinterval_t ch_delta_list::delta`

Time delta before timeout.
8.3 ch_dyn_element Struct Reference

Type of a dynamic object list element.

#include <chfactory.h>

Collaboration diagram for ch_dyn_element:

![Collaboration Diagram]

Data Fields

- struct ch_dyn_element * next
  
  Next dynamic object in the list.

- ucnt_t refs
  
  Number of references to this object.

8.3.1 Detailed Description

Type of a dynamic object list element.

8.3.2 Field Documentation

8.3.2.1 next

struct ch_dyn_element* ch_dyn_element::next

Next dynamic object in the list.

8.3.2.2 refs

ucnt_t ch_dyn_element::refs

Number of references to this object.
8.4 ch_dyn_list Struct Reference

Type of a dynamic object list.

#include <chfactory.h>

Collaboration diagram for ch_dyn_list:

8.4.1 Detailed Description

Type of a dynamic object list.

8.5 ch_dyn_mailbox Struct Reference

Type of a dynamic buffer object.

#include <chfactory.h>
Collaboration diagram for ch_dyn_mailbox:

Data Fields

- `dyn_element_t element`
  
  List element of the dynamic buffer object.

- `mailbox_t mbx`
  
  The mailbox.

8.5.1 Detailed Description

Type of a dynamic buffer object.

8.5.2 Field Documentation
8.5.2.1 element

dyn_element_t ch_dyn_mailbox::element

List element of the dynamic buffer object.

8.5.2.2 mbx

mailbox_t ch_dyn_mailbox::mbx

The mailbox.

8.6 ch_dyn_object Struct Reference

Type of a dynamic buffer object.

#include <chfactory.h>

Collaboration diagram for ch_dyn_object:

![Collaboration diagram for ch_dyn_object](image)

**Data Fields**

- dyn_element_t element

List element of the dynamic buffer object.
8.6.1 Detailed Description

Type of a dynamic buffer object.

8.6.2 Field Documentation

8.6.2.1 element

dyn_element_t ch_dyn_object::element

List element of the dynamic buffer object.

8.7 ch_dyn_objects_fifo Struct Reference

Type of a dynamic buffer object.

#include <chfactory.h>

Collaboration diagram for ch_dyn_objects_fifo:
Data Fields

- **dyn_element_t element**
  
  List element of the dynamic buffer object.

- **objects_fifo_t fifo**
  
  The objects FIFO.

### 8.7.1 Detailed Description

Type of a dynamic buffer object.

### 8.7.2 Field Documentation

#### 8.7.2.1 element

\[
\text{dyn_element_t ch_dyn_objects_fifo::element}
\]

List element of the dynamic buffer object.

#### 8.7.2.2 fifo

\[
\text{objects_fifo_t ch_dyn_objects_fifo::fifo}
\]

The objects FIFO.

### 8.8 ch_dyn_pipe Struct Reference

Type of a dynamic pipe object.

```
#include <chfactory.h>
```
Collaboration diagram for ch_dyn_pipe:

Data Fields

- **dyn_element_t element**
  
  *List element of the dynamic pipe object.*

- **pipe_t pipe**
  
  *The pipe.*
8.8.1 Detailed Description

Type of a dynamic pipe object.

8.8.2 Field Documentation

8.8.2.1 element

dyn_element_t ch_dyn_pipe::element

List element of the dynamic pipe object.

8.8.2.2 pipe

pipe_t ch_dyn_pipe::pipe

The pipe.

8.9 ch_dyn_semaphore Struct Reference

Type of a dynamic semaphore.

#include <chfactory.h>

Collaboration diagram for ch_dyn_semaphore:
Data Fields

- `dyn_element_t element`
  List element of the dynamic semaphore.
- `semaphore_t sem`
  The semaphore.

8.9.1 Detailed Description

Type of a dynamic semaphore.

8.9.2 Field Documentation

8.9.2.1 element

dyn_element_t ch_dyn_semaphore::element

List element of the dynamic semaphore.

8.9.2.2 sem

semaphore_t ch_dyn_semaphore::sem

The semaphore.

8.10 ch_job_descriptor Struct Reference

Type of a job descriptor.

#include <chjobs.h>

Collaboration diagram for ch_job_descriptor:
8.11 ch_jobs_queue Struct Reference

Data Fields

- job_function_t jobfunc
  
  Job function.

- void * jobarg
  
  Argument to be passed to the job function.

8.10.1 Detailed Description

Type of a job descriptor.

8.10.2 Field Documentation

8.10.2.1 jobfunc

job_function_t ch_job_descriptor::jobfunc

Job function.

8.10.2.2 jobarg

void* ch_job_descriptor::jobarg

Argument to be passed to the job function.

8.11 ch_jobs_queue Struct Reference

Type of a jobs queue.

#include <chjobs.h>

ChibiOS/RT
Collaboration diagram for ch_jobs_queue:

Data Fields

- guarded_memory_pool_t free
  
  Pool of the free jobs.
- mailbox_t mbx
  
  Mailbox of the sent jobs.

8.11.1 Detailed Description

Type of a jobs queue.

8.11.2 Field Documentation
### 8.12 ch_list Struct Reference

Structure representing a generic single link list header and element.

```c
#include <chlists.h>
```

Collaboration diagram for ch_list:

![Collaboration Diagram](image-url)

#### Data Fields

- `ch_list_t * next`

  *Next in the list/queue.*

#### 8.12.1 Detailed Description

Structure representing a generic single link list header and element.

#### 8.12.2 Field Documentation
8.12.2.1 next

`ch_list_t* ch_list::next`

Next in the list/queue.

8.13 ch_mutex Struct Reference

Mutex structure.

```
#include <chmtx.h>
```

Collaboration diagram for ch_mutex:

![Collaboration diagram]

Data Fields

- **ch_queue_t queue**
  
  *Queue of the threads sleeping on this mutex.*

- **thread_t * owner**
  
  *Owner thread_t pointer or NULL.*

- **mutex_t * next**
  
  *Next mutex_t into an owner-list or NULL.*

- **cnt_t cnt**
  
  *Mutex recursion counter.*
8.13 ch_mutex Struct Reference

8.13.1 Detailed Description

Mutex structure.

8.13.2 Field Documentation

8.13.2.1 queue

ch_queue_t ch_mutex::queue

Queue of the threads sleeping on this mutex.

8.13.2.2 owner

thread_t* ch_mutex::owner

Owner thread_t pointer or NULL.

8.13.2.3 next

mutex_t* ch_mutex::next

Next mutex_t into an owner-list or NULL.

8.13.2.4 cnt

cnt_t ch_mutex::cnt

Mutex recursion counter.
8.14 ch_objects_cache Struct Reference

Structure representing a cache object.

#include <chobjcaches.h>

Collaboration diagram for ch_objects_cache:
Data Fields

- ucnt_t hashn
  Number of elements in the hash table.
- oc_hash_header_t * hashp
  Pointer to the hash table.
- ucnt_t objn
  Number of elements in the objects table.
- size_t objsz
  Size of elements in the objects table.
- void * objvp
  Pointer to the objects table.
- oc_lru_header_t lru
  LRU list header.
- semaphore_t cache_sem
  Semaphore for cache access.
- semaphore_t lru_sem
  Semaphore for LRU access.
- oc_readf_t readf
  Reader functions for cached objects.
- oc_writef_t writef
  Writer functions for cached objects.

8.14.1 Detailed Description

Structure representing a cache object.

8.14.2 Field Documentation

8.14.2.1 hashn

ucnt_t ch_objects_cache::hashn

Number of elements in the hash table.

8.14.2.2 hashp

oc_hash_header_t * ch_objects_cache::hashp

Pointer to the hash table.
8.14.2.3 objn

`ucnt_t ch_objects_cache::objn`

Number of elements in the objects table.

8.14.2.4 objsz

`size_t ch_objects_cache::objsz`

Size of elements in the objects table.

8.14.2.5 objvp

`void* ch_objects_cache::objvp`

Pointer to the objects table.

8.14.2.6 lru

`oc_lru_header_t ch_objects_cache::lru`

LRU list header.

8.14.2.7 cache_sem

`semaphore_t ch_objects_cache::cache_sem`

Semaphore for cache access.

8.14.2.8 lru_sem

`semaphore_t ch_objects_cache::lru_sem`

Semaphore for LRU access.
8.15.2.9 readf

`oc_readf_t ch_objects_cache::readf`

Reader functions for cached objects.

8.15.2.10 writef

`oc_writef_t ch_objects_cache::writef`

Writer functions for cached objects.

8.15 ch_objects_factory Struct Reference

Type of the factory main object.

`#include <chfactory.h>`

Collaboration diagram for ch_objects_factory:
Data Fields

- **mutex_t mtx**
  Factory access mutex or semaphore.

- **dyn_list_t obj_list**
  List of the registered objects.

- **memory_pool_t obj_pool**
  Pool of the available registered objects.

- **dyn_list_t buf_list**
  List of the allocated buffer objects.

- **dyn_list_t sem_list**
  List of the allocated semaphores.

- **memory_pool_t sem_pool**
  Pool of the available semaphores.

- **dyn_list_t mbx_list**
  List of the allocated buffer objects.

- **dyn_list_t fifo_list**
  List of the allocated "objects FIFO" objects.

- **dyn_list_t pipe_list**
  List of the allocated pipe objects.

8.15.1 Detailed Description

Type of the factory main object.

8.15.2 Field Documentation

8.15.2.1 mtx

```c
mutex_t ch_objects_factory::mtx
```

Factory access mutex or semaphore.

8.15.2.2 obj_list

```c
dyn_list_t ch_objects_factory::obj_list
```

List of the registered objects.
8.15.2.3  obj_pool

memory_pool_t ch_objects_factory::obj_pool

Pool of the available registered objects.

8.15.2.4  buf_list

dyn_list_t ch_objects_factory::buf_list

List of the allocated buffer objects.

8.15.2.5  sem_list

dyn_list_t ch_objects_factory::sem_list

List of the allocated semaphores.

8.15.2.6  sem_pool

memory_pool_t ch_objects_factory::sem_pool

Pool of the available semaphores.

8.15.2.7  mbx_list

dyn_list_t ch_objects_factory::mbx_list

List of the allocated buffer objects.

8.15.2.8  fifo_list

dyn_list_t ch_objects_factory::fifo_list

List of the allocated "objects FIFO" objects.
8.15.2.9 pipe_list

dyn_list_t ch_objects_factory::pipe_list

List of the allocated pipe objects.

8.16 ch_objects_fifo Struct Reference

Type of an objects FIFO.

#include <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.
8.16.1 Detailed Description

Type of an objects FIFO.

8.16.2 Field Documentation

8.16.2.1 free

\texttt{guarded_memory_pool_t} ch_objects_fifo::free

Pool of the free objects.

8.16.2.2 mbx

\texttt{mailbox_t} ch_objects_fifo::mbx

Mailbox of the sent objects.

8.17 ch_oc_hash_header Struct Reference

Structure representing an hash table element.

\texttt{#include <chobjcaches.h>
Collaboration diagram for ch_oc_hash_header:

Data Fields

- oc_object_t * hash_next
  
  Next in the collisions list.

- oc_object_t * hash_prev
  
  Previous in the collisions list.

8.17.1 Detailed Description

Structure representing an hash table element.

8.17.2 Field Documentation
8.17.2.1 hash_next

\texttt{oc\_object\_t* ch\_oc\_hash\_header::hash\_next}

Next in the collisions list.

8.17.2.2 hash_prev

\texttt{oc\_object\_t* ch\_oc\_hash\_header::hash\_prev}

Previous in the collisions list.

8.18 ch\_oc\_lru\_header Struct Reference

Structure representing an hash table element.

\texttt{#include <chobjcaches.h>}

Collaboration diagram for ch\_oc\_lru\_header:
Data Fields

- `oc_object_t * hash_next`
  Next in the collisions list.
- `oc_object_t * hash_prev`
  Previous in the collisions list.
- `oc_object_t * lru_next`
  Next in the LRU list.
- `oc_object_t * lru_prev`
  Previous in the LRU list.

8.18.1 Detailed Description

Structure representing an hash table element.

8.18.2 Field Documentation

8.18.2.1 hash_next

`oc_object_t* ch_oc_lru_header::hash_next`

Next in the collisions list.

8.18.2.2 hash_prev

`oc_object_t* ch_oc_lru_header::hash_prev`

Previous in the collisions list.

8.18.2.3 lru_next

`oc_object_t* ch_oc_lru_header::lru_next`

Next in the LRU list.

8.18.2.4 lru_prev

`oc_object_t* ch_oc_lru_header::lru_prev`

Previous in the LRU list.
8.19 ch_oc_object Struct Reference

Structure representing a cached object.

#include <chobjcaches.h>

Collaboration diagram for ch_oc_object:

Data Fields

- oc_object_t * hash_next
  Next in the collisions list.
- oc_object_t * hash_prev
  Previous in the collisions list.
- oc_object_t * lru_next
  Next in the LRU list.
- oc_object_t * lru_prev
  Previous in the LRU list.
- uint32_t obj_group
  Object group.
- uint32_t obj_key
  Object key.
- semaphore_t obj_sem
Semaphore for object access.

- oc_flags_t obj_flags
  Object flags.
- void * dptr
  User pointer.

8.19.1 Detailed Description

Structure representing a cached object.

8.19.2 Field Documentation

8.19.2.1 hash_next

oc_object_t* ch_oc_object::hash_next

Next in the collisions list.

8.19.2.2 hash_prev

oc_object_t* ch_oc_object::hash_prev

Previous in the collisions list.

8.19.2.3 lru_next

oc_object_t* ch_oc_object::lru_next

Next in the LRU list.

8.19.2.4 lru_prev

oc_object_t* ch_oc_object::lru_prev

Previous in the LRU list.
8.19.2.5 obj_group

uint32_t ch_oc_object::obj_group

Object group.

8.19.2.6 obj_key

uint32_t ch_oc_object::obj_key

Object key.

8.19.2.7 obj_sem

semaphore_t ch_oc_object::obj_sem

Semaphore for object access.

8.19.2.8 obj_flags

oc_flags_t ch_oc_object::obj_flags

Object flags.

8.19.2.9 dptr

void* ch_oc_object::dptr

User pointer.

Note

This pointer can be used to refer to external buffers, chCacheObjectInit() initializes it to NULL.
8.20 ch_priority_queue Struct Reference

Structure representing a generic priority-ordered bidirectional linked list header and element.

```c
#include <chlists.h>
```

Collaboration diagram for ch_priority_queue:

![Collaboration Diagram]

**Data Fields**

- `ch_priority_queue_t * next`
  
  Next in the list/queue.

- `ch_priority_queue_t * prev`
  
  Previous in the queue.

### 8.20.1 Detailed Description

Structure representing a generic priority-ordered bidirectional linked list header and element.

**Note**

Link fields are void pointers in order to avoid aliasing issues.

### 8.20.2 Field Documentation

#### 8.20.2.1 next

```c
ch_priority_queue_t* ch_priority_queue::next
```

Next in the list/queue.
8.20.2.2 prev

\code{ch_priority_queue_t* ch_priority_queue::prev}

Previous in the queue.

8.21 ch_queue Struct Reference

Structure representing a generic bidirectional linked list header and element.

\texttt{#include <chlists.h>}

Collaboration diagram for \texttt{ch_queue}:

\begin{center}
\includegraphics[width=0.3\textwidth]{ch_queue_diagram.png}
\end{center}

Data Fields

- \texttt{ch_queue_t * next}
  
  Next in the list/queue.

- \texttt{ch_queue_t * prev}
  
  Previous in the queue.

8.21.1 Detailed Description

Structure representing a generic bidirectional linked list header and element.

8.21.2 Field Documentation
8.21.2.1 next

`ch_queue_t* ch_queue::next`

Next in the list/queue.

8.21.2.2 prev

`ch_queue_t* ch_queue::prev`

Previous in the queue.

8.22 ch_registered_static_object Struct Reference

Type of a registered object.

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

Collaboration diagram for `ch_registered_static_object`:

```
Data Fields

- `dyn_element_t element`
  List element of the registered object.
- `void * objp`
  Pointer to the object.
```
8.22.1 Detailed Description

Type of a registered object.

8.22.2 Field Documentation

8.22.2.1 element

`dyn_element_t ch_registered_static_object::element`

List element of the registered object.

8.22.2.2 objp

`void* ch_registered_static_object::objp`

Pointer to the object.

Note

The type of the object is not stored in anyway.

8.23 ch_semaphore Struct Reference

Semaphore structure.

```
#include <chsem.h>
```

Inheritance diagram for ch_semaphore:

```
ChibiOS/RT
```
Collaboration diagram for ch_semaphore:

Data Fields

- ch_queue_t queue
  Queue of the threads sleeping on this semaphore.

- cnt_t cnt
  The semaphore counter.

8.23.1 Detailed Description

Semaphore structure.

8.23.2 Field Documentation

8.23.2.1 queue

ch_queue_t ch_semaphore::queue

Queue of the threads sleeping on this semaphore.
8.23.2.2 cnt

cnt_t ch_semaphore::cnt

The semaphore counter.

8.24 ch_system Struct Reference

System data structure.

#include <chschd.h>

Collaboration diagram for ch_system:

Data Fields

- ready_list_t rlist
  
  Ready list header.

- virtual_timers_list_t vtlist
  
  Virtual timers delta list header.

- system_debug_t dbg
  
  System debug.

- thread_t mainthread
  
  Main thread descriptor.

- tm_calibration_t tm
  
  Time measurement calibration data.

- kernel_stats_t kernel_stats
  
  Global kernel statistics.
8.24.1 Detailed Description

System data structure.

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

8.24.2 Field Documentation

8.24.2.1 rlist

\texttt{ready\_list\_t ch\_system::rlist}

Ready list header.

8.24.2.2 vtlist

\texttt{virtual\_timers\_list\_t ch\_system::vtlist}

Virtual timers delta list header.

8.24.2.3 dbg

\texttt{system\_debug\_t ch\_system::dbg}

System debug.

8.24.2.4 mainthread

\texttt{thread\_t ch\_system::mainthread}

Main thread descriptor.
8.24.2.5 tm

tm_calibration_t ch_system::tm

Time measurement calibration data.

8.24.2.6 kernel_stats

kernel_stats_t ch_system::kernel_stats

Global kernel statistics.

8.25 ch_system_debug Struct Reference

System debug data structure.

#include <chschd.h>
Data Fields

- **const char volatile** panic_msg
  
  Pointer to the panic message.
- cnt_t isr_cnt
  
  ISR nesting level.
- cnt_t lock_cnt
  
  Lock nesting level.
- ch_trace_buffer_t trace_buffer
  
  Public trace buffer.
8.25 ch_system_debug Struct Reference

8.25.1 Detailed Description

System debug data structure.

8.25.2 Field Documentation

8.25.2.1 panic_msg

const char* volatile ch_system_debug::panic_msg

Pointer to the panic message.

This pointer is meant to be accessed through the debugger, it is written once and then the system is halted.

Note

Accesses to this pointer must never be optimized out so the field itself is declared volatile.

8.25.2.2 isr_cnt

cnt_t ch_system_debug::isr_cnt

ISR nesting level.

8.25.2.3 lock_cnt

cnt_t ch_system_debug::lock_cnt

Lock nesting level.

8.25.2.4 trace_buffer

ch_trace_buffer_t ch_system_debug::trace_buffer

Public trace buffer.
8.26 ch_thread Struct Reference

Structure representing a thread.

#include <chschd.h>

Collaboration diagram for ch_thread:

Data Fields

- struct port_context ctx
  
  Processor context.

- thread_t * newer
  
  Newer registry element.

- thread_t * older
  
  Older registry element.

- const char * name
  
  Thread name or NULL.

- stkalign_t * wabase
  
  Working area base address.

- tstate_t state
  
  Thread state.
Current thread state.

- **tmode_t flags**
  Various thread flags.
- **trefs_t refs**
  References to this thread.
- **tslices_t ticks**
  Number of ticks remaining to this thread.
- **volatile systime_t time**
  Thread consumed time in ticks.

- **union {**
  - **msg_t rdymsg**
    Thread wakeup code.
  - **msg_t exitcode**
    Thread exit code.
  - **void * wtobjp**
    Pointer to a generic "wait" object.
  - **thread_reference_t * wttrp**
    Pointer to a generic thread reference object.
  - **msg_t sentmsg**
    Thread sent message.
  - **struct ch_semaphore * wtsemp**
    Pointer to a generic semaphore object.
  - **struct ch_mutex * wtmtxp**
    Pointer to a generic mutex object.
  - **eventmask_t ewmask**
    Enabled events mask.
}

State-specific fields.

- **ch_list_t waiting**
  Termination waiting list.
- **ch_queue_t msgqueue**
  Messages queue.
- **eventmask_t epending**
  Pending events mask.
- **struct ch_mutex * mtxlist**
  List of the mutexes owned by this thread.
- **tprio_t realprio**
  Thread's own, non-inherited, priority.
- **void * mpool**
  Memory Pool where the thread workspace is returned.
- **time_measurement_t stats**
  Thread statistics.
- **ch_list_t list**
  Threads lists element.

- **ch_queue_t queue**
  Threads queues element.

- **ch_priority_queue_t pqueue**
  Threads ordered queues element.
8.26.1 Detailed Description

Structure representing a thread.

Note

Not all the listed fields are always needed, by switching off some not needed ChibiOS/RT subsystems it is possible to save RAM space by shrinking this structure.

8.26.2 Field Documentation

8.26.2.1 list

\texttt{ch\_list\_t ch\_thread::list}

Threads lists element.

8.26.2.2 queue

\texttt{ch\_queue\_t ch\_thread::queue}

Threads queues element.

8.26.2.3 pqueue

\texttt{ch\_priority\_queue\_t ch\_thread::pqueue}

Threads ordered queues element.

8.26.2.4 ctx

\texttt{struct port\_context ch\_thread::ctx}

Processor context.
### 8.26.2.5 newer

```c
thread_t* ch_thread::newer
```

Newer registry element.

### 8.26.2.6 older

```c
thread_t* ch_thread::older
```

Older registry element.

### 8.26.2.7 name

```c
const char* ch_thread::name
```

Thread name or NULL.

### 8.26.2.8 wabase

```c
stkalign_t* ch_thread::wabase
```

Working area base address.

**Note**

This pointer is used for stack overflow checks and for dynamic threading.

### 8.26.2.9 state

```c
tstate_t ch_thread::state
```

Current thread state.
8.26.2.10  flags

\texttt{tmode_t ch\_thread::flags}

Various thread flags.

8.26.2.11  refs

\texttt{trefs_t ch\_thread::refs}

References to this thread.

8.26.2.12  ticks

\texttt{tslices_t ch\_thread::ticks}

Number of ticks remaining to this thread.

8.26.2.13  time

\texttt{volatile systime_t ch\_thread::time}

Thread consumed time in ticks.

\textbf{Note}

This field can overflow.

8.26.2.14  rdymsg

\texttt{msg_t ch\_thread::rdymsg}

Thread wakeup code.

\textbf{Note}

This field contains the low level message sent to the thread by the waking thread or interrupt handler. The value is valid after exiting the \texttt{chSchWakeupS()} function.
8.26.2.15  exitcode

`msg_t ch_thread::exitcode`

Thread exit code.

**Note**
The thread termination code is stored in this field in order to be retrieved by the thread performing a `chThdWait()` on this thread.

8.26.2.16  wtobjp

`void* ch_thread::wtobjp`

Pointer to a generic "wait" object.

**Note**
This field is used to get a generic pointer to a synchronization object and is valid when the thread is in one of the wait states.

8.26.2.17  wttrp

`thread_reference_t* ch_thread::wttrp`

Pointer to a generic thread reference object.

**Note**
This field is used to get a pointer to a synchronization object and is valid when the thread is in `CH_STATE_SUSPENDED` state.

8.26.2.18  sentmsg

`msg_t ch_thread::sentmsg`

Thread sent message.
8.26.2.19 wtsemp

`struct ch_semaphore* ch_thread::wtsemp`

Pointer to a generic semaphore object.

Note

This field is used to get a pointer to a synchronization object and is valid when the thread is in CH_STATE←_WTSEM state.

8.26.2.20 wtmtxp

`struct ch_mutex* ch_thread::wtmtxp`

Pointer to a generic mutex object.

Note

This field is used to get a pointer to a synchronization object and is valid when the thread is in CH_STATE←_WTMTX state.

8.26.2.21 ewmask

`eventmask_t ch_thread::ewmask`

Enabled events mask.

Note

This field is only valid while the thread is in the CH_STATE_WTOREVT or CH_STATE_WTANDEVT states.

8.26.2.22 u

`union { ... } ch_thread::u`

State-specific fields.

Note

All the fields declared in this union are only valid in the specified state or condition and are thus volatile.
8.26.2.23 waiting

`ch_list_t ch_thread::waiting`
Termination waiting list.

8.26.2.24 msgqueue

`ch_queue_t ch_thread::msgqueue`
Messages queue.

8.26.2.25 epending

`eventmask_t ch_thread::epending`
Pending events mask.

8.26.2.26 mtxlist

`struct ch_mutex* ch_thread::mtxlist`
List of the mutexes owned by this thread.

Note

The list is terminated by a `NULL` in this field.

8.26.2.27 realprio

`tprio_t ch_thread::reaprio`
Thread's own, non-inherited, priority.

8.26.2.28 mpool

`void* ch_thread::mpool`
Memory Pool where the thread workspace is returned.
8.26.2.29 stats

time_measurement_t ch_thread::stats

Thread statistics.

8.27 ch_threads_queue Struct Reference

Structure representing a threads queue.

#include <chsched.h>

Inherited by ch_ready_list.

Collaboration diagram for ch_threads_queue:

Data Fields

• ch_queue_t queue

  Threads queue header.

8.27.1 Detailed Description

Structure representing a threads queue.

8.27.2 Field Documentation
8.27.2.1 queue

`ch_queue_t ch_threads_queue::queue`

Threads queue header.

---

8.28 ch_trace_buffer_t Struct Reference

Trace buffer header.

```c
#include <chtrace.h>
```
Collaboration diagram for `ch_trace_buffer_t`:

```
+ suspended
+ size
```

```
ch_trace_event_t
+ type
+ state
+ rtstamp
+ time
+ wtobjp
+ sw
+ ... name
+ wabase
+ state
+ flags
+ refs
+ ticks
+ time
+ rdymsg
and 8 more...
```

```
+buffer
+ptr
```

```
+ntp
```

Data Fields

- `uint16_t suspended`
  
  Suspended trace sources mask.

- `uint16_t size`
  
  Trace buffer size (entries).

- `ch_trace_event_t * ptr`
8.28 ch_trace_buffer_t Struct Reference

8.28.1 Detailed Description

Trace buffer header.

8.28.2 Field Documentation

8.28.2.1 suspended

```c
uint16_t ch_trace_buffer_t::suspended
```

Suspended trace sources mask.

8.28.2.2 size

```c
uint16_t ch_trace_buffer_t::size
```

Trace buffer size (entries).

8.28.2.3 ptr

```c
ch_trace_event_t* ch_trace_buffer_t::ptr
```

Pointer to the buffer front.

8.28.2.4 buffer

```c
ch_trace_event_t ch_trace_buffer_t::buffer[CH_DBG_TRACE_BUFFER_SIZE]
```

Ring buffer.
8.29  ch_trace_event_t Struct Reference

Trace buffer record.

#include <chtrace.h>

Collaboration diagram for ch_trace_event_t:

```
Data Fields

- uint32_t type:3
  Record type.
- uint32_t state:5
  Switched out thread state.
- uint32_t rtstamp:24
  Accurate time stamp.
- systime_t time
  System time stamp of the switch event.
- thread_t * ntp
  Switched in thread.
- void * wtobjp
  ChibiOS/RT
```
Object where going to sleep.

- struct {
  thread_t ∗ ntp
  Switched in thread.
  void ∗ wtobjp
  Object where going to sleep.
} sw

Structure representing a context switch.

- const char ∗ name
  ISR function name taken using func.

- struct {
  const char ∗ name
  ISR function name taken using func.
} isr

Structure representing an ISR enter.

- const char ∗ reason
  Halt error string.

- struct {
  const char ∗ reason
  Halt error string.
} halt

Structure representing an halt.

- void ∗ up1
  Trace user parameter 1.
- void ∗ up2
  Trace user parameter 2.

- struct {
  void ∗ up1
  Trace user parameter 1.
  void ∗ up2
  Trace user parameter 2.
} user

User trace structure.

### 8.29.1 Detailed Description

Trace buffer record.

### 8.29.2 Field Documentation

#### 8.29.2.1 type

uint32_t ch_trace_event_t::type

Record type.
8.29.2.2  state

uint32_t ch_trace_event_t::state

Switched out thread state.

8.29.2.3  rtstamp

uint32_t ch_trace_event_t::rtstamp

Accurate time stamp.

Note

This field only available if the post supports PORT_SUPPORTS_RT else it is set to zero.

8.29.2.4  time

systime_t ch_trace_event_t::time

System time stamp of the switch event.

8.29.2.5  ntp

thread_t* ch_trace_event_t::ntp

Switched in thread.

8.29.2.6  wtobjp

void* ch_trace_event_t::wtobjp

Object where going to sleep.

8.29.2.7  sw

struct [ ... ] ch_trace_event_t::sw

Structure representing a context switch.
8.29.2.8  name

const char* ch_trace_event_t::name

ISR function name taken using func.

8.29.2.9  isr

struct { ... } ch_trace_event_t::isr

Structure representing an ISR enter.

8.29.2.10 reason

const char* ch_trace_event_t::reason

Halt error string.

8.29.2.11 halt

struct { ... } ch_trace_event_t::halt

Structure representing an halt.

8.29.2.12 up1

void* ch_trace_event_t::up1

Trace user parameter 1.

8.29.2.13 up2

void* ch_trace_event_t::up2

Trace user parameter 2.
8.29.2.14 user

struct {...} ch_trace_event_t::user

User trace structure.

8.30 ch_virtual_timer Struct Reference

Structure representing a Virtual Timer.

#include <chsched.h>

Collaboration diagram for ch_virtual_timer:

Data Fields

- delta_list_t dlist
  Delta list element.

- vfunc_t func
  Timer callback function pointer.

- void * par
  Timer callback function parameter.

8.30.1 Detailed Description

Structure representing a Virtual Timer.
8.30.2 Field Documentation

8.30.2.1 dlist

delta_list_t ch_virtual_timer::dlist
Delta list element.

8.30.2.2 func

vtfunc_t ch_virtual_timer::func
Timer callback function pointer.

8.30.2.3 par

void* ch_virtual_timer::par
Timer callback function parameter.

8.31 ch_virtual_timers_list Struct Reference

Structure representing a virtual timers list header.

#include <chschd.h>

Collaboration diagram for ch_virtual_timers_list:
Data Fields

- `delta_list_t dlist`
  Delta list header.

- `volatile systime_t systime`
  System Time counter.

- `systime_t lasttime`
  System time of the last tick event.

8.31.1 Detailed Description

Structure representing a virtual timers list header.

Note

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

8.31.2 Field Documentation

8.31.2.1 `dlist`

`delta_list_t ch_virtual_timers_list::dlist`

Delta list header.

8.31.2.2 `systime`

`volatile systime_t ch_virtual_timers_list::systime`

System Time counter.

8.31.2.3 `lasttime`

`systime_t ch_virtual_timers_list::lasttime`

System time of the last tick event.

System time of the last tick event.
8.32 chdebug_t Struct Reference

ChibiOS/RT memory signature record.

#include <chregistry.h>

Collaboration diagram for chdebug_t:

<table>
<thead>
<tr>
<th>chdebug_t</th>
</tr>
</thead>
</table>
| + identifier 
| + zero |
| + size |
| + version |
| + ptrsize |
| + timesize |
| + threadsize |
| + off_prio |
| + off_ctx |
| + off_newer |
| and 8 more... |

Data Fields

- `char identifier [4]
  Always set to "main".`
- `uint8_t zero
  Must be zero.`
- `uint8_t size
  Size of this structure.`
- `uint16_t version
  Encoded ChibiOS/RT version.`
- `uint8_t ptrsize
  Size of a pointer.`
- `uint8_t timesize
  Size of a systime_t.`
- `uint8_t threadsize
  Size of a thread_t.`
- `uint8_t off_prio
  Offset of prio field.`
- `uint8_t off_ctx
  `
8.32 Data Structure Documentation

8.32.1 Detailed Description

ChibiOS/RT memory signature record.

8.32.2 Field Documentation

8.32.2.1 identifier

char chdebug_t::identifier[4]

Always set to "main".

8.32.2.2 zero

uint8_t chdebug_t::zero

Must be zero.
8.32.2.3 size

uint8_t chdebug_t::size

Size of this structure.

8.32.2.4 version

uint16_t chdebug_t::version

Encoded ChibiOS/RT version.

8.32.2.5 ptrsize

uint8_t chdebug_t::ptrsize

Size of a pointer.

8.32.2.6 timesize

uint8_t chdebug_t::timesize

Size of a systime_t.

8.32.2.7 threadsize

uint8_t chdebug_t::threadsize

Size of a thread_t.

8.32.2.8 off_prio

uint8_t chdebug_t::off_prio

Offset of prio field.
8.32.2.9  off_ctx

```c
uint8_t chdebug_t::off_ctx
```
Offset of `ctx` field.

8.32.2.10  off_newer

```c
uint8_t chdebug_t::off_newer
```
Offset of `newer` field.

8.32.2.11  off_older

```c
uint8_t chdebug_t::off_older
```
Offset of `older` field.

8.32.2.12  off_name

```c
uint8_t chdebug_t::off_name
```
Offset of `name` field.

8.32.2.13  off_stklimit

```c
uint8_t chdebug_t::off_stklimit
```
Offset of `stklimit` field.

8.32.2.14  off_state

```c
uint8_t chdebug_t::off_state
```
Offset of `state` field.
8.32.15 off_flags

```c
uint8_t chdebug_t::off_flags
```
Offset of flags field.

8.32.16 off.refs

```c
uint8_t chdebug_t::off.refs
```
Offset of refs field.

8.32.17 off.preempt

```c
uint8_t chdebug_t::off.preempt
```
Offset of preempt field.

8.32.18 off.time

```c
uint8_t chdebug_t::off.time
```
Offset of time field.

8.33 condition_variable Struct Reference

condition_variable_t structure.

```c
#include <chcond.h>
```
Collaboration diagram for condition_variable:

![Collaboration Diagram](image-url)
Data Fields

- \texttt{ch_queue_t queue}
  
  Condition variable threads queue.

8.33.1 Detailed Description

\texttt{condition_variable_t} structure.

8.33.2 Field Documentation

8.33.2.1 queue

\texttt{ch_queue_t condition_variable::queue}

Condition variable threads queue.

8.34 event_listener Struct Reference

Event Listener structure.

#include <chevents.h>

Collaboration diagram for \texttt{event_listener}:
Data Fields

- `event_listener_t * next`
  Next Event Listener registered on the event source.

- `thread_t * listener`
  Thread interested in the event source.

- `eventmask_t events`
  Events to be set in the listening thread.

- `eventflags_t flags`
  Flags added to the listener by the event source.

- `eventflags_t wflags`
  Flags that this listener interested in.

8.34.1 Detailed Description

Event Listener structure.

8.34.2 Field Documentation

8.34.2.1 next

`event_listener_t* event_listener::next`

Next Event Listener registered on the event source.

8.34.2.2 listener

`thread_t* event_listener::listener`

Thread interested in the event source.

8.34.2.3 events

`eventmask_t event_listener::events`

Events to be set in the listening thread.
8.34.2.4 flags

```c
eventflags_t event_listener::flags
```

Flags added to the listener by the event source.

8.34.2.5 wflags

```c
eventflags_t event_listener::wflags
```

Flags that this listener interested in.

8.35 event_source Struct Reference

Event Source structure.

```c
#include <chevents.h>
```
8.35 event_source Struct Reference

Collaboration diagram for event_source:

Data Fields

- `event_listener_t * next`

  First Event Listener registered on the Event Source.

8.35.1 Detailed Description

Event Source structure.

8.35.2 Field Documentation
8.35.2.1  next

event_listener_t* event_source::next

First Event Listener registered on the Event Source.

8.36  guarded_memory_pool_t Struct Reference

Guarded memory pool descriptor.

#include <chmempools.h>

Collaboration diagram for guarded_memory_pool_t:

Data Fields

- semaphore_t sem
  Counter semaphore guarding the memory pool.

- memory_pool_t pool
  The memory pool itself.
8.36.1 Detailed Description

Guarded memory pool descriptor.

8.36.2 Field Documentation

8.36.2.1 sem

`semaphore_t guarded_memory_pool_t::sem`

Counter semaphore guarding the memory pool.

8.36.2.2 pool

`memory_pool_t guarded_memory_pool_t::pool`

The memory pool itself.

8.37 heap_header Union Reference

Memory heap block header.

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

Collaboration diagram for heap_header:
8.37.1 Detailed Description

Memory heap block header.

8.37.2 Field Documentation

8.37.2.1 next

heap_header_t* heap_header::next

Next block in free list.

8.37.2.2 pages

size_t heap_header::pages

Size of the area in pages.

8.37.2.3 heap

memory_heap_t* heap_header::heap

Block owner heap.

8.37.2.4 size

size_t heap_header::size

Size of the area in bytes.
8.38 kernel_stats_t Struct Reference

Type of a kernel statistics structure.

#include <chstats.h>

Collaboration diagram for kernel_stats_t:

![Collaboration Diagram]

Data Fields

- `ucnt_t n_irq`
  
  Number of IRQs.

- `ucnt_t n_ctxswc`
  
  Number of context switches.

- `time_measurement_t m_crit_thd`
  
  Measurement of threads critical zones duration.

- `time_measurement_t m_crit_isr`
  
  Measurement of ISRs critical zones duration.

8.38.1 Detailed Description

Type of a kernel statistics structure.
8.38.2 Field Documentation

8.38.2.1 n_irq

`ucnt_t kernel_stats_t::n_irq`

Number of IRQs.

8.38.2.2 n_ctxswc

`ucnt_t kernel_stats_t::n_ctxswc`

Number of context switches.

8.38.2.3 m_crit_thd

`time_measurement_t kernel_stats_t::m_crit_thd`

Measurement of threads critical zones duration.

8.38.2.4 m_crit_isr

`time_measurement_t kernel_stats_t::m_crit_isr`

Measurement of ISRs critical zones duration.
8.39 mailbox_t Struct Reference

Structure representing a mailbox object.

```
#include <chmboxes.h>
```

Collaboration diagram for mailbox_t:

![Collaboration diagram](image)

**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.
8.39.1 Detailed Description

Structure representing a mailbox object.

8.39.2 Field Documentation

8.39.2.1 buffer

```
msg_t* mailbox_t::buffer
```

Pointer to the mailbox buffer.

8.39.2.2 top

```
msg_t* mailbox_t::top
```

Pointer to the location after the buffer.

8.39.2.3 wrptr

```
msg_t* mailbox_t::wrptr
```

Write pointer.
8.39.2.4 rdptr

```c
msg_t* mailbox_t::rdptr
```

Read pointer.

8.39.2.5 cnt

```c
size_t mailbox_t::cnt
```

Messages in queue.

8.39.2.6 reset

```c
bool mailbox_t::reset
```

True in reset state.

8.39.2.7 qw

```c
threads_queue_t mailbox_t::qw
```

Queued writers.

8.39.2.8 qr

```c
threads_queue_t mailbox_t::qr
```

Queued readers.
8.40  memcore_t Struct Reference

Type of memory core object.

#include <chmemcore.h>

Collaboration diagram for memcore_t:

Data Fields

- `uint8_t * basemem`
  
  Next free address.

- `uint8_t * topmem`
  
  Final address.

8.40.1  Detailed Description

Type of memory core object.

8.40.2  Field Documentation

8.40.2.1  basemem

`uint8_t* memcore_t::basemem`

Next free address.

8.40.2.2  topmem

`uint8_t* memcore_t::topmem`

Final address.
8.41 memory_heap Struct Reference

Structure describing a memory heap.

#include <chmemheaps.h>

Collaboration diagram for memory_heap:
Data Fields

- **memgetfunc2_t provider**
  Memory blocks provider for this heap.

- **heap_header_t header**
  Free blocks list header.

- **mutex_t mtx**
  Heap access mutex.

### 8.41.1 Detailed Description

Structure describing a memory heap.

### 8.41.2 Field Documentation

#### 8.41.2.1 provider

```c
memgetfunc2_t memory_heap::provider
```

Memory blocks provider for this heap.

#### 8.41.2.2 header

```c
heap_header_t memory_heap::header
```

Free blocks list header.

#### 8.41.2.3 mtx

```c
mutex_t memory_heap::mtx
```

Heap access mutex.
8.42 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.

8.42.1 Detailed Description

Memory pool descriptor.

8.42.2 Field Documentation
8.42.1 next

struct pool_header* memory_pool_t::next

Pointer to the header.

8.42.2 object_size

size_t memory_pool_t::object_size

Memory pool objects size.

8.42.3 align

unsigned memory_pool_t::align

Required alignment.

8.42.4 provider

memgetfunc_t memory_pool_t::provider

Memory blocks provider for this pool.

8.43 pipe_t Struct Reference

Structure representing a pipe object.

#include <chpipes.h>
Collaboration diagram for `pipe_t`:

```
Data Fields

- `uint8_t * buffer`
  Pointer to the pipe buffer.

- `uint8_t * top`
  Pointer to the location after the buffer.

- `uint8_t * wrptr`
  Write pointer.

- `uint8_t * rdptr`
  Read pointer.

- `size_t cnt`
  Bytes in the pipe.

- `bool reset`
```

[ChibiOS/RT]
True if in reset state.
• `thread_reference_t wtr`
  Waiting writer.
• `thread_reference_t rtr`
  Waiting reader.
• `mutex_t cmtx`
  Common access mutex.
• `mutex_t wmtx`
  Write access mutex.
• `mutex_t rmtx`
  Read access mutex.

### 8.43.1 Detailed Description

Structure representing a pipe object.

### 8.43.2 Field Documentation

#### 8.43.2.1 buffer

```c
uint8_t* pipe_t::buffer
```

Pointer to the pipe buffer.

#### 8.43.2.2 top

```c
uint8_t* pipe_t::top
```

Pointer to the location after the buffer.

#### 8.43.2.3 wrptr

```c
uint8_t* pipe_t::wrptr
```

Write pointer.
8.43.2.4 rdptr

uint8_t* pipe_t::rdptr

Read pointer.

8.43.2.5 cnt

size_t pipe_t::cnt

Bytes in the pipe.

8.43.2.6 reset

bool pipe_t::reset

True if in reset state.

8.43.2.7 wtr

thread_reference_t pipe_t::wtr

Waiting writer.

8.43.2.8 rtr

thread_reference_t pipe_t::rtr

Waiting reader.

8.43.2.9 cmtx

mutex_t pipe_t::cmtx

Common access mutex.
8.43.2.10  wmtx

mutex_t pipe_t::wmtx

Write access mutex.

8.43.2.11  rmtx

mutex_t pipe_t::rmtx

Read access mutex.

8.44  pool_header Struct Reference

Memory pool free object header.

#include <chmempools.h>

Collaboration diagram for pool_header:

![Collaboration Diagram](image)

Data Fields

- struct pool_header * next

  Pointer to the next pool header in the list.

8.44.1  Detailed Description

Memory pool free object header.
8.44.2 Field Documentation

8.44.2.1 next

struct pool_header* pool_header::next

Pointer to the next pool header in the list.

8.45 thread_descriptor_t Struct Reference

Type of a thread descriptor.

#include <chthreads.h>

Collaboration diagram for thread_descriptor_t:

```
<table>
<thead>
<tr>
<th>thread_descriptor_t</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ name</td>
</tr>
<tr>
<td>+ wbase</td>
</tr>
<tr>
<td>+ wend</td>
</tr>
<tr>
<td>+ prio</td>
</tr>
<tr>
<td>+ funcp</td>
</tr>
<tr>
<td>+ arg</td>
</tr>
</tbody>
</table>
```

Data Fields

- const char * name
  
  Thread name.
- stkalign_t * wbase
  
  Pointer to the working area base.
- stkalign_t * wend
  
  End of the working area.
- tprio_t prio
  
  Thread priority.
- tfunc_t funcp
  
  Thread function pointer.
- void * arg
  
  Thread argument.
8.45.1 Detailed Description

Type of a thread descriptor.

8.45.2 Field Documentation

8.45.2.1 name

`const char* thread_descriptor_t::name`

Thread name.

8.45.2.2 wbase

`stkalign_t* thread_descriptor_t::wbase`

Pointer to the working area base.

8.45.2.3 wend

`stkalign_t* thread_descriptor_t::wend`

End of the working area.

8.45.2.4 prio

`t prio_t thread_descriptor_t::prio`

Thread priority.

8.45.2.5 funcp

`tfunc_t thread_descriptor_t::funcp`

Thread function pointer.
8.45.2.6  arg

void* thread_descriptor_t::arg

Thread argument.

8.46  time_measurement_t Struct Reference

Type of a Time Measurement object.

#include <chtm.h>

Collaboration diagram for time_measurement_t:

```
+ best
+ worst
+ last
+ n
+ cumulative
```

Data Fields

- rtcnt_t best
  
  Best measurement.

- rtcnt_t worst
  
  Worst measurement.

- rtcnt_t last
  
  Last measurement.

- ucnt_t n
  
  Number of measurements.

- rttime_t cumulative
  
  Cumulative measurement.
8.46.1 Detailed Description

Type of a Time Measurement object.

Note
The maximum measurable time period depends on the implementation of the realtime counter and its clock frequency.
The measurement is not 100% cycle-accurate, it can be in excess of few cycles depending on the compiler and target architecture.
Interrupts can affect measurement if the measurement is performed with interrupts enabled.

8.46.2 Field Documentation

8.46.2.1 best

```
rtcnt_t time_measurement_t::best
```

Best measurement.

8.46.2.2 worst

```
rtcnt_t time_measurement_t::worst
```

Worst measurement.

8.46.2.3 last

```
rtcnt_t time_measurement_t::last
```

Last measurement.

8.46.2.4 n

```
ucnt_t time_measurement_t::n
```

Number of measurements.
8.46.2.5 cumulative

```c
rttime_t time_measurement_t::cumulative
```
Cumulative measurement.

8.47 tm_calibration_t Struct Reference

Type of a time measurement calibration data.

```c
#include <chtm.h>
```

Collaboration diagram for tm_calibration_t:

```
+ offset
```

Data Fields

- rtcnt_t offset
  
  Measurement calibration value.

8.47.1 Detailed Description

Type of a time measurement calibration data.

8.47.2 Field Documentation

8.47.2.1 offset

```c
rtcnt_t tm_calibration_t::offset
```
Measurement calibration value.
Chapter 9

File Documentation

9.1 ch.h File Reference

ChibiOS/RT main include file.

```
#include "chconf.h"
#include "chchecks.h"
#include "chlicense.h"
#include "chrestrictions.h"
#include "ctypes.h"
#include "chsystypes.h"
#include "chlists.h"
#include "chdebug.h"
#include "ctime.h"
#include "chalign.h"
#include "chcore.h"
#include "chtrace.h"
#include "chtm.h"
#include "chstats.h"
#include "chsched.h"
#include "chsys.h"
#include "chvt.h"
#include "chthreads.h"
#include "chregistry.h"
#include "chsem.h"
#include "chmtx.h"
#include "chcond.h"
#include "chevents.h"
#include "chmsg.h"
#include "chlib.h"
#include "chdynamic.h"
```

Macros

- `#define _CHIBIOS_RT_`
  ChibiOS/RT identification macro.
- `#define CH_KERNEL_STABLE 1`
Stable release flag.

ChibiOS/RT version identification

- `#define CH_KERNEL_VERSION "6.1.4"`
  Kernel version string.
- `#define CH_KERNEL_MAJOR 6`
  Kernel version major number.
- `#define CH_KERNEL_MINOR 1`
  Kernel version minor number.
- `#define CH_KERNEL_PATCH 4`
  Kernel version patch number.

Constants for configuration options

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

Functions

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

9.1.1 Detailed Description

ChibiOS/RT main include file.

9.2 chalign.h File Reference

Memory alignment macros and structures.

Macros

Memory alignment support macros

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

Memory alignment macros and structures.

9.3 chbsem.h File Reference

Binary semaphores structures and macros.

Data Structures

- struct ch_binary_semaphore
  
  Binary semaphore type.

Macros

- #define _BSEMAPHORE_DATA(name, taken) (_SEMAPHORE_DATA(name.sem, ((taken) ? 0 : 1)))
  
  Data part of a static semaphore initializer.

- #define BSEMAPHORE_DECL(name, taken) binary_semaphore_t name = _BSEMAPHORE_DATA(name, taken)
  
  Static semaphore initializer.

Typedefs

- typedef struct ch_binary_semaphore binary_semaphore_t

  Binary semaphore type.

Functions

- static void chBSemObjectInit (binary_semaphore_t *bsp, bool taken)
  
  Initializes a binary semaphore.

- static msg_t chBSemWait (binary_semaphore_t *bsp)
  
  Wait operation on the binary semaphore.

- static msg_t chBSemWaitS (binary_semaphore_t *bsp)
  
  Wait operation on the binary semaphore.

- static msg_t chBSemWaitTimeoutS (binary_semaphore_t *bsp, sysinterval_t timeout)
  
  Wait operation on the binary semaphore.

- static msg_t chBSemWaitTimeout (binary_semaphore_t *bsp, sysinterval_t timeout)
  
  Wait operation on the binary semaphore.

- static void 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.
9.3.1 Detailed Description

Binary semaphores structures and macros.

Binary semaphores related APIs and services.

Operation mode

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

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

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

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

9.4 chchecks.h File Reference

Configuration file checks header.

9.4.1 Detailed Description

Configuration file checks header.

9.5 chcond.c File Reference

Condition Variables code.

#include "ch.h"
Functions

- void chCondObjectInit (condition_variable_t *cp)
  Initializes a condition_variable_t structure.
- void chCondSignal (condition_variable_t *cp)
  Signals one thread that is waiting on the condition variable.
- void chCondSignalAll (condition_variable_t *cp)
  Signals one thread that is waiting on the condition variable.
- void chCondBroadcast (condition_variable_t *cp)
  Signals all threads that are waiting on the condition variable.
- void chCondBroadcastI (condition_variable_t *cp)
  Signals all threads that are waiting on the condition variable.
- msg_t chCondWait (condition_variable_t *cp)
  Waits on the condition variable releasing the mutex lock.
- msg_t chCondWaitS (condition_variable_t *cp)
  Waits on the condition variable releasing the mutex lock.
- msg_t chCondWaitTimeout (condition_variable_t *cp, sysinterval_t timeout)
  Waits on the condition variable releasing the mutex lock.
- msg_t chCondWaitTimeoutS (condition_variable_t *cp, sysinterval_t timeout)
  Waits on the condition variable releasing the mutex lock.

9.5.1 Detailed Description

Condition Variables code.

9.6 chcond.h File Reference

Condition Variables macros and structures.

Data Structures

- struct condition_variable
  condition_variable_t structure.

Macros

- #define _CONDVAR_DATA(name) { _CH_QUEUE_DATA(name.queue)}
  Data part of a static condition variable initializer.
- #define CONDVARDECL(name) condition_variable_t name = _CONDVAR_DATA(name)
  Static condition variable initializer.

Typedefs

- typedef struct condition_variable condition_variable_t
  condition_variable_t structure.
Functions

- void chCondObjectInit (condition_variable_t *cp)
  Initializes a condition_variable_t structure.
- void chCondSignal (condition_variable_t *cp)
  Signals one thread that is waiting on the condition variable.
- void chCondSignalAll (condition_variable_t *cp)
  Signals one thread that is waiting on the condition variable.
- void chCondBroadcast (condition_variable_t *cp)
  Signals all threads that are waiting on the condition variable.
- void chCondBroadcastI (condition_variable_t *cp)
  Signals all threads that are waiting on the condition variable.
- msg_t chCondWait (condition_variable_t *cp)
  Waits on the condition variable releasing the mutex lock.
- msg_t chCondWaitS (condition_variable_t *cp)
  Waits on the condition variable releasing the mutex lock.
- msg_t chCondWaitTimeout (condition_variable_t *cp, sysinterval_t timeout)
  Waits on the condition variable releasing the mutex lock.
- msg_t chCondWaitTimeoutS (condition_variable_t *cp, sysinterval_t timeout)
  Waits on the condition variable releasing the mutex lock.

9.6.1 Detailed Description

Condition Variables macros and structures.

9.7 chconf.h File Reference

Configuration file template.

Macros

System timers settings

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

Kernel parameters and options

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

Performance options

- `#define CH_CFG_OPTIMIZE_SPEED TRUE`
  OS optimization.

Subsystem options

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

OSLIB options

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

Jobs Queues APIs.

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

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

Kernel hooks
- \#define CH_CFG_SYSTEM_EXTRA_FIELDS /* Add threads custom fields here.*/
  System structure extension.
- \#define CH_CFG_SYSTEM_INIT_HOOK()
  System initialization hook.
- \#define CH_CFG_THREAD_EXTRA_FIELDS /* Add threads custom fields here.*/
  Threads descriptor structure extension.
- \#define CH_CFG_THREAD_INIT_HOOK(tp)
  Threads initialization hook.
- \#define CH_CFG_THREAD_EXIT_HOOK(tp)
  Threads finalization hook.
- \#define CH_CFG_CONTEXT SWITCH_HOOK(ntp, otp)
Context switch hook.
- \#define CH_CFG_IRQ_PROLOGUE_HOOK()
  ISR enter hook.
- \#define CH_CFG_IRQ_EPILOGUE_HOOK()
  ISR exit hook.
- \#define CH_CFG_IDLE_ENTER_HOOK()
  Idle thread enter hook.
- \#define CH_CFG_IDLE_LEAVE_HOOK()
  Idle thread leave hook.
- \#define CH_CFG_IDLE_LOOP_HOOK()
  Idle Loop hook.
- \#define CH_CFG_SYSTEM_TICK_HOOK()
  System tick event hook.
- \#define CH_CFG_SYSTEM_HALT_HOOK(reason)
  System halt hook.
- \#define CH_CFG_TRACE_HOOK(tep)
  Trace hook.

9.7.1 Detailed Description

Configuration file template.

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

9.8 chcustomer.h File Reference

Customer-related info.

Macros

- \#define CH_CUSTOMER_ID_STRING "Santa, North Pole"
  Customer readable identifier.
- \#define CH_CUSTOMER_ID_CODE "xxxx-yyyy"
  Customer code.
- \#define CH_LICENSE_LINKLIB RCMP
  Current license.

Licensed Products

- \#define CH_CUSTOMER_LIC_RT TRUE
- \#define CH_CUSTOMER_LIC_NIL TRUE
- \#define CH_CUSTOMER_LIC_OSLIB TRUE
- \#define CH_CUSTOMER_LIC_EX TRUE
- \#define CH_CUSTOMER_LIC_SB TRUE
- \#define CH_CUSTOMER_LIC_PORT_CM0 TRUE
- \#define CH_CUSTOMER_LIC_PORT_CM3 TRUE
- \#define CH_CUSTOMER_LIC_PORT_CM4 TRUE
- \#define CH_CUSTOMER_LIC_PORT_CM7 TRUE
- \#define CH_CUSTOMER_LIC_PORT_ARM79 TRUE
- \#define CH_CUSTOMER_LIC_PORT_E200Z0 TRUE
- \#define CH_CUSTOMER_LIC_PORT_E200Z2 TRUE
- \#define CH_CUSTOMER_LIC_PORT_E200Z3 TRUE
- \#define CH_CUSTOMER_LIC_PORT_E200Z4 TRUE
9.8.1 Detailed Description

Customer-related info.

9.9 chdebug.c File Reference

Debug support code.

#include "ch.h"

Functions

• void _dbg_check_disable (void)
  Guard code for chSysDisable().
• void _dbg_check_suspend (void)
  Guard code for chSysSuspend().
• void _dbg_check_enable (void)
  Guard code for chSysEnable().
• void _dbg_check_lock (void)
  Guard code for chSysLock().
• void _dbg_check_unlock (void)
  Guard code for chSysUnlock().
• void _dbg_check_lock_from_isr (void)
  Guard code for chSysLockFromIsr().
• void _dbg_check_unlock_from_isr (void)
  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.

9.9.1 Detailed Description

Debug support code.

9.10 chdebug.h File Reference

Debug support macros and structures.
Macros

Debug related settings

- #define CH_DBG_STACK_FILL_VALUE 0x55
  Fill value for thread stack area in debug mode.

Macro Functions

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

9.10.1 Detailed Description

Debug support macros and structures.

9.11 chdelegates.c File Reference

Delegate threads code.

#include "ch.h"

Functions

- msg_t __ch_delegate_fn0 (va_list *argsp)
  Veneer for functions with no parameters.
- msg_t __ch_delegate_fn1 (va_list *argsp)
  Veneer for functions with one parameter.
- msg_t __ch_delegate_fn2 (va_list *argsp)
  Veneer for functions with two parameters.
- msg_t __ch_delegate_fn3 (va_list *argsp)
  Veneer for functions with three parameters.
- msg_t __ch_delegate_fn4 (va_list *argsp)
  Veneer for functions with four parameters.
- msg_t chDelegateCallVeneer (thread_t *tp, delegate_veneer_t veneer,...)
  Triggers a function call on a delegate thread.
- void chDelegateDispatch (void)
  Call messages dispatching.
- msg_t chDelegateDispatchTimeout (sysinterval_t timeout)
  Call messages dispatching with timeout.

9.11.1 Detailed Description

Delegate threads code.

Delegate threads.
Operation mode

A delegate thread is a thread performing function calls triggered by other threads. This functionality is especially useful when encapsulating a library not designed for threading into a delegate thread. Other threads have access to the library without having to worry about mutual exclusion.

Precondition

In order to use the pipes APIs the CH_CFG_USE_DELEGATES option must be enabled in chconf.h.

Note

Compatible with RT and NIL.

9.12 chdelegates.h File Reference

Delegate threads macros and structures.

#include <stdarg.h>

Typedefs

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

Functions

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

9.12.1 Detailed Description
Delegate threads macros and structures.

9.13 chdynamic.c File Reference
Dynamic threads code.
#include "ch.h"

Functions
• thread_t * chThdCreateFromHeap (memory_heap_t *heapp, size_t size, const char *name, tprio_t prio, tfunc_t pf, void *arg)
  Creates a new thread allocating the memory from the heap.
• thread_t * chThdCreateFromMemoryPool (memory_pool_t *mp, const char *name, tprio_t prio, tfunc_t pf, void *arg)
  Creates a new thread allocating the memory from the specified memory pool.

9.13.1 Detailed Description
Dynamic threads code.

9.14 chdynamic.h File Reference
Dynamic threads macros and structures.
Functions

- thread_t * chThdCreateFromHeap (memory_heap_t *heapp, size_t size, const char *name, tprio_t prio, tfunc_t pf, void *arg)
  
  Creates a new thread allocating the memory from the heap.

- thread_t * chThdCreateFromMemoryPool (memory_pool_t *mp, const char *name, tprio_t prio, tfunc_t pf, void *arg)
  
  Creates a new thread allocating the memory from the specified memory pool.

9.14.1 Detailed Description

Dynamic threads macros and structures.

9.15 chevents.c File Reference

Events code.

#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 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.

- 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 chEvtWaitOne (eventmask_t events)`
  Waits for exactly one of the specified events.
- `eventmask_t chEvtWaitAny (eventmask_t events)`
  Waits for any of the specified events.
- `eventmask_t chEvtWaitAll (eventmask_t events)`
  Waits for all the specified events.
- `eventmask_t chEvtWaitOneTimeout (eventmask_t events, sysinterval_t timeout)`
  Waits for exactly one of the specified events.
- `eventmask_t chEvtWaitAnyTimeout (eventmask_t events, sysinterval_t timeout)`
  Waits for any of the specified events.
- `eventmask_t chEvtWaitAllTimeout (eventmask_t events, sysinterval_t timeout)`
  Waits for all the specified events.

### 9.15.1 Detailed Description

Events code.

### 9.16 chevents.h File Reference

Events macros and structures.

#### Data Structures

- `struct event_listener`
  Event Listener structure.
- `struct event_source`
  Event Source structure.

#### Macros

- `#define ALL_EVENTS ((eventmask_t)-1)`
  All events allowed mask.
- `#define EVENT_MASK(eid) ((eventmask_t)(1) << (eventmask_t)(eid))`
  Returns an event mask from an event identifier.
- `#define _EVENTSOURCE_DATA(name) {(event_listener_t ∗)(&name)}`
  Data part of a static event source initializer.
- `#define EVENTSOURCE_DECL(name) event_source_t name = _EVENTSOURCE_DATA(name)`
  Static event source initializer.

#### Typedefs

- `typedef struct event_source event_source_t`
  Event Source structure.
- `typedef void (∗evhandler_t)(eventid_t id)`
  Event Handler callback function.
Functions

- void 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 chEvtWaitOne (eventmask_t events)
  Waits for exactly one of the specified events.

- eventmask_t chEvtWaitAny (eventmask_t events)
  Waits for any of the specified events.

- eventmask_t chEvtWaitAll (eventmask_t events)
  Waits for all the specified events.

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

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

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

- static void chEvtObjectInit (event_source_t *esp)
  Initializes an Event Source.

- static void chEvtRegisterMask (event_source_t *esp, event_listener_t *elp, eventmask_t events)
  Registers an Event Listener on an Event Source.

- static void chEvtRegister (event_source_t *esp, event_listener_t *elp, eventid_t event)
  Registers an Event Listener on an Event Source.

- static bool chEvtIsListeningI (event_source_t *esp)
  Verifies if there is at least one event_listener_t registered.

- static void chEvtBroadcast (event_source_t *esp)
  Signals all the Event Listeners registered on the specified Event Source.

- static void chEvtBroadcastI (event_source_t *esp)
Signals all the Event Listeners registered on the specified Event Source.

- static eventmask_t chEvtAddEventsI (eventmask_t events)
  Adds (OR) a set of events to the current thread, this is much faster than using chEvtBroadcast() or chEvtSignal().
- static eventmask_t chEvtGetEventsX (void)
  Returns the events mask.

9.16.1 Detailed Description

Events macros and structures.

9.17 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.
- 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 * chFactoryFindObjectsFIFO (const char *name)`
  
  Retrieves a dynamic "objects FIFO" object.

- `void chFactoryReleaseObjectsFIFO (dyn_objects_fifo_t *dofp)`
  
  Releases a dynamic "objects FIFO" object.

- `dyn_pipe_t * chFactoryCreatePipe (const char *name, size_t size)`
  
  Creates a dynamic pipe object.

- `dyn_pipe_t * chFactoryFindPipe (const char *name)`
  
  Retrieves a dynamic pipe object.

- `void chFactoryReleasePipe (dyn_pipe_t *dpp)`
  
  Releases a dynamic pipe object.

Variables

- `objects_factory_t ch_factory`
  
  Factory object static instance.

9.17.1 Detailed Description

ChibiOS objects factory and registry code.

9.18 chfactory.h File Reference

ChibiOS objects factory structures and macros.

Data Structures

- `struct ch_dyn_element`
  
  Type of a dynamic object list element.

- `struct ch_dyn_list`
  
  Type of a dynamic object list.

- `struct ch_registered_static_object`
  
  Type of a registered object.

- `struct ch_dyn_object`
  
  Type of a dynamic buffer object.

- `struct ch_dyn_semaphore`
  
  Type of a dynamic semaphore.

- `struct ch_dyn_mailbox`
  
  Type of a dynamic buffer object.

- `struct ch_dyn_objects_fifo`
  
  Type of a dynamic buffer object.

- `struct ch_dyn_pipe`
  
  Type of a dynamic pipe object.

- `struct ch_objects_factory`
  
  Type of the factory main object.
Macros

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

Typedefs

- `typedef struct ch_dyn_element dyn_element_t`
  Type of a dynamic object list element.
- `typedef struct ch_dyn_list dyn_list_t`
  Type of a dynamic object list.
- `typedef struct ch_registered_static_object registered_object_t`
  Type of a registered object.
- `typedef struct ch_dyn_buffer 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 mailbox.
- `typedef struct ch_dyn_objects_fifo dyn_objects_fifo_t`
  Type of a dynamic buffer object.
- `typedef struct ch_dyn_pipe dyn_pipe_t`
  Type of a dynamic pipe object.
- `typedef struct ch_objects_factory objects_factory_t`
  Type of the factory main object.
Functions

- void _factory_init (void)
  Initializes the objects factory.
- registered_object_t * chFactoryRegisterObject (const char *name, void *objp)
  Registers a generic object.
- registered_object_t * chFactoryFindObject (const char *name)
  Retrieves a registered object.
- registered_object_t * chFactoryFindObjectByPointer (void *objp)
  Retrieves a registered object by pointer.
- void chFactoryReleaseObject (registered_object_t *rop)
  Releases a registered object.
- dyn_buffer_t * chFactoryCreateBuffer (const char *name, size_t size)
  Creates a generic dynamic buffer object.
- dyn_buffer_t * chFactoryFindBuffer (const char *name)
  Retrieves a dynamic buffer object.
- void chFactoryReleaseBuffer (dyn_buffer_t *dbp)
  Releases a dynamic buffer object.
- dyn_semaphore_t * chFactoryCreateSemaphore (const char *name, cnt_t n)
  Creates a dynamic semaphore object.
- dyn_semaphore_t * chFactoryFindSemaphore (const char *name)
  Retrieves a dynamic semaphore object.
- void chFactoryReleaseSemaphore (dyn_semaphore_t *dsp)
  Releases a dynamic semaphore object.
- dyn_mailbox_t * chFactoryCreateMailbox (const char *name, size_t n)
  Creates a dynamic mailbox object.
- dyn_mailbox_t * chFactoryFindMailbox (const char *name)
  Retrieves a dynamic mailbox object.
- void chFactoryReleaseMailbox (dyn_mailbox_t *dmp)
  Releases a dynamic mailbox object.
- dyn_objects_fifo_t * chFactoryCreateObjectsFIFO (const char *name, size_t objsize, size_t objn, unsigned objalign)
  Creates a dynamic "objects FIFO" object.
- dyn_objects_fifo_t * chFactoryFindObjectByPointer (void *objp)
  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 semaphore_t * chFactoryGetSemaphore (dyn_semaphore_t *dsp)
  Returns the pointer to the inner semaphore.
• static mailbox_t * chFactoryGetMailbox (dyn_mailbox_t *dmp)
  Returns the pointer to the inner mailbox.
• static objects_fifo_t * chFactoryGetObjectsFIFO (dyn_objects_fifo_t *dofp)
  Returns the pointer to the inner objects FIFO.
• static pipe_t * chFactoryGetPipe (dyn_pipe_t *dpp)
  Returns the pointer to the inner pipe.

9.18.1 Detailed Description

ChibiOS objects factory structures and macros.

9.19 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 * chJobGetl (jobs_queue_t *jqp)
  
  Allocates a free job object.

- static job_descriptor_t * chJobGetTimeoutS (jobs_queue_t *jqp, sysinterval_t timeout)
  
  Allocates a free job object.

- static job_descriptor_t * chJobGetTimeout (jobs_queue_t *jqp, sysinterval_t timeout)
  
  Allocates a free job object.

- static void 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.

9.19.1 Detailed Description

Jobs Queues structures and macros.

This module implements queues of generic jobs to be delegated asynchronously to a pool of dedicated threads. Operations defined for Jobs Queues

- **Get**: An job object is taken from the pool of the available jobs.
- **Post**: A job is posted to the queue, it will be returned to the pool after execution.

9.20 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 1
  Stable release flag.

ChibiOS/LIB version identification

- #define CH_OSLIB_VERSION "1.2.1"
  OS Library version string.
- #define CH_OSLIB_MAJOR 1
  OS Library version major number.
- #define CH_OSLIB_MINOR 2
  OS Library version minor number.
- #define CH_OSLIB_PATCH 1
  OS Library version patch number.

Functions

- static void _oslib_init (void)
  Initialization of all library modules.

9.20.1 Detailed Description

ChibiOS/LIB main include file.

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

9.21 chlicense.h File Reference

License Module macros and structures.

#include "chcustomer.h"
Macros

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

Allowed Features Levels

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

Deployment Options

- \#define CH_DEPLOY_UNLIMITED -1
- \#define CH_DEPLOY_NONE 0

Licensing Options

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

9.21.1 Detailed Description

License Module macros and structures.

9.22 chlists.h File Reference

Lists and Queues header.

Data Structures

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

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

Typedefs

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

Functions

- `static void ch_list_init (ch_list_t* lp)`
  List initialization.
- `static bool ch_list_isempty (ch_list_t* lp)`
  Evaluates to true if the specified list is empty.
- `static bool ch_list_notempty (ch_list_t* lp)`
  Evaluates to true if the specified list is not empty.
- `static void ch_list_push (ch_list_t* p, ch_list_t* lp)`
  Pushes an element on top of a stack list.
- `static ch_list_t* ch_list_pop (ch_list_t* lp)`
  Pops an element from the top of a stack list and returns it.
- `static void ch_queue_init (ch_queue_t* qp)`
  Queue initialization.
- `static bool ch_queue_isempty (const ch_queue_t* qp)`
  Evaluates to true if the specified queue is empty.
- `static bool ch_queue_notempty (const ch_queue_t* qp)`
  Evaluates to true if the specified queue is not empty.
- `static void ch_queue_insert (ch_queue_t* p, ch_queue_t* qp)`
  Inserts an element into a queue.
- `static ch_queue_t* ch_queue_fifo_remove (ch_queue_t* qp)`
  Removes the first-out element from a queue and returns it.
- `static ch_queue_t* ch_queue_lifo_remove (ch_queue_t* qp)`
  Removes the last-out element from a queue and returns it.
- `static ch_queue_t* ch_queue_dequeue (ch_queue_t* p)`
  Removes an element from a queue and returns it.
- `static void ch_pqueue_init (ch_priority_queue_t* pqp)`
  Priority queue initialization.
- `static ch_priority_queue_t* ch_pqueue_remove_highest (ch_priority_queue_t* pqp)`
  Removes the highest priority element from a priority queue and returns it.
- `static ch_priority_queue_t* ch_pqueue_insert_behind (ch_priority_queue_t* pqp, ch_priority_queue_t* p)`
  Inserts an element in the priority queue placing it behind its peers.
- `static ch_priority_queue_t* ch_pqueue_insert_ahead (ch_priority_queue_t* pqp, ch_priority_queue_t* p)`
  Inserts an element in the priority queue placing it ahead of its peers.
9.22.1 Detailed Description

Lists and Queues header.

9.23 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.*

9.23.1 Detailed Description

Mailboxes code.

9.24 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 chMBGetSizel (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.
9.24.1 Detailed Description

Mailboxes macros and structures.

9.25 chmemcore.c File Reference

Core memory manager code.

#include "ch.h"

Functions

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

- void * chCoreAllocFromBaseI (size_t size, unsigned align, size_t offset)
  
  Allocates a memory block starting from the lowest address upward.

- void * chCoreAllocFromTopI (size_t size, unsigned align, size_t offset)
  
  Allocates a memory block starting from the top address downward.

- void * chCoreAllocFromBase (size_t size, unsigned align, size_t offset)
  
  Allocates a memory block starting from the lowest address upward.

- void * chCoreAllocFromTop (size_t size, unsigned align, size_t offset)
  
  Allocates a memory block starting from the top address downward.

- size_t chCoreGetStatusX (void)
  
  Core memory status.

Variables

- memcore_t ch_memcore
  
  Memory core descriptor.

9.25.1 Detailed Description

Core memory manager code.

9.26 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 chCoreAllocAlignedWithOffset 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.

9.26.1 Detailed Description

Core memory manager macros and structures.

9.27 chmemheaps.c File Reference

Memory heaps code.

#include "ch.h"
Functions

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

Variables

- **static** memory_heap_t default_heap
  
  Default heap descriptor.

9.27.1 Detailed Description

Memory heaps code.

9.28 chmemheaps.h File Reference

Memory heaps macros and structures.

Data Structures

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

Macros

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

Typedefs

- **typedef** struct memory_heap memory_heap_t
  
  Type of a memory heap.
- **typedef** union heap_header heap_header_t
  
  Type of a memory heap header.
Functions

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

9.28.1 Detailed Description

Memory heaps macros and structures.

9.29 chmempools.c File Reference

Memory Pools code.

#include "ch.h"

Functions

- void chPoolObjectInitAligned (memory_pool_t *mp, size_t size, unsigned align, memgetfunc_t provider)
  Initializes an empty memory pool.
- void chPoolLoadArray (memory_pool_t *mp, void *p, size_t n)
  Loads a memory pool with an array of static objects.
- void * chPoolAlloc (memory_pool_t *mp)
  Allocates an object from a memory pool.
- void * chPoolAlloc (memory_pool_t *mp)
  Allocates an object from a memory pool.
- void chPoolFree (memory_pool_t *mp, void *objp)
  Releases an object into a memory pool.
- void chPoolFree (memory_pool_t *mp, void *objp)
  Releases an object into a memory pool.
- void chGuardedPoolObjectInitAligned (guarded_memory_pool_t *gmp, size_t size, unsigned align)
  Initializes an empty guarded memory pool.
- void chGuardedPoolLoadArray (guarded_memory_pool_t *gmp, void *p, size_t n)
  Loads a guarded memory pool with an array of static objects.
- void * chGuardedPoolAllocTimeoutS (guarded_memory_pool_t *gmp, sysinterval_t timeout)
  Allocates an object from a guarded memory pool.
- void * chGuardedPoolAllocTimeout (guarded_memory_pool_t *gmp, sysinterval_t timeout)
  Allocates an object from a guarded memory pool.
- void chGuardedPoolFree (guarded_memory_pool_t *gmp, void *objp)
  Releases an object into a guarded memory pool.
9.29.1 Detailed Description

Memory Pools code.

9.30 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)`
  ChibiOS/RT
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.

### 9.30.1 Detailed Description

Memory Pools macros and structures.

### 9.31 chmsg.c File Reference

Messages code.

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

#### Functions

- `msg_t chMsgSend (thread_t *tp, msg_t msg)`
  Sends a message to the specified thread.

- `thread_t * chMsgWaitS (void)`
 Suspends the thread and waits for an incoming message.

- `thread_t * chMsgWaitTimeoutS (sysinterval_t timeout)`
  Suspends the thread and waits for an incoming message or a timeout to occur.

- `thread_t * chMsgPollS (void)`
  Poll to check for an incoming message.

- `void chMsgRelease (thread_t *tp, msg_t msg)`
  Releases a sender thread specifying a response message.
9.31.1 Detailed Description

Messages code.

9.32 chmsg.h File Reference

Messages macros and structures.

Functions

- **msg_t chMsgSend (thread_t *tp, msg_t msg)**
  Sends a message to the specified thread.
- **thread_t * chMsgWaitS (void)**
  Suspends the thread and waits for an incoming message.
- **thread_t * chMsgWaitTimeoutS (sysinterval_t timeout)**
  Suspends the thread and waits for an incoming message or a timeout to occur.
- **thread_t * chMsgPollS (void)**
  Poll to check for an incoming message.
- **void chMsgRelease (thread_t *tp, msg_t msg)**
  Releases a sender thread specifying a response message.
- **static thread_t * chMsgWait (void)**
  Suspends the thread and waits for an incoming message.
- **static thread_t * chMsgWaitTimeout (sysinterval_t timeout)**
  Suspends the thread and waits for an incoming message or a timeout to occur.
- **static thread_t * chMsgPoll (void)**
  Poll to check for an incoming message.
- **static bool chMsgIsPendingI (thread_t *tp)**
  Evaluates to true if the thread has pending messages.
- **static msg_t chMsgGet (thread_t *tp)**
  Returns the message carried by the specified thread.
- **static void chMsgReleaseS (thread_t *tp, msg_t msg)**
  Releases the thread waiting on top of the messages queue.

9.32.1 Detailed Description

Messages macros and structures.

9.33 chmtx.c File Reference

Mutexes code.

#include "ch.h"
Functions

- void chMtxObjectInit (mutex_t *mp)
  Initializes a mutex_t structure.
- void chMtxLock (mutex_t *mp)
  Locks the specified mutex.
- void chMtxLockS (mutex_t *mp)
  Locks the specified mutex.
- bool chMtxTryLock (mutex_t *mp)
  Tries to lock a mutex.
- bool chMtxTryLockS (mutex_t *mp)
  Tries to lock a mutex.
- void chMtxUnlock (mutex_t *mp)
  Unlocks the specified mutex.
- void chMtxUnlockS (mutex_t *mp)
  Unlocks the specified mutex.
- void chMtxUnlockAllS (void)
  Unlocks all mutexes owned by the invoking thread.
- void chMtxUnlockAll (void)
  Unlocks all mutexes owned by the invoking thread.

9.33.1 Detailed Description

Mutexes code.

9.34 chmtx.h File Reference

Mutexes macros and structures.

Data Structures

- struct ch_mutex
  Mutex structure.

Macros

- #define _MUTEX_DATA(name) { _CH_QUEUE_DATA(name.queue), NULL, NULL, 0 }
  Data part of a static mutex initializer.
- #define MUTEX_DECL(name) mutex_t name = _MUTEX_DATA(name)
  Static mutex initializer.

Typedefs

- typedef struct ch_mutex mutex_t
  Type of a mutex structure.
Functions

- void chMtxObjectInit (mutex_t *mp)
  Initializes a mutex_t structure.
- void chMtxLock (mutex_t *mp)
  Locks the specified mutex.
- void chMtxLockS (mutex_t *mp)
  Locks the specified mutex.
- bool chMtxTryLock (mutex_t *mp)
  Tries to lock a mutex.
- bool chMtxTryLockS (mutex_t *mp)
  Tries to lock a mutex.
- void chMtxUnlock (mutex_t *mp)
  Unlocks the specified mutex.
- void chMtxUnlockS (mutex_t *mp)
  Unlocks the specified mutex.
- void chMtxUnlockAll (void)
  Unlocks all mutexes owned by the invoking thread.
- void chMtxUnlockAllS (void)
  Unlocks all mutexes owned by the invoking thread.
- static bool chMtxQueueNotEmptyS (mutex_t *mp)
  Returns true if the mutex queue contains at least a waiting thread.
- static thread_t * chMtxGetOwnerI (mutex_t *mp)
  Returns the mutex owner thread.
- static mutex_t * chMtxGetNextMutexX (void)
  Returns the next mutex in the mutexes stack of the current thread.

9.34.1 Detailed Description

Mutexes macros and structures.

9.35 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 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 (object_cache_t *ocp, oc_object_t *objp)

Releases an object into the cache.

- bool chCacheReadStream (object_cache_t *ocp, oc_object_t *objp, bool async)

Reads object data from the storage.

- bool chCacheWriteObject (object_cache_t *ocp, oc_object_t *objp, bool async)

Writes the object data back to storage.

### 9.35.1 Detailed Description

Objects Caches code.

Objects caches.

#### Operation mode

An object cache allows to retrieve and release objects from a slow media, for example a disk or flash. The most recently used objects are kept in a series of RAM buffers making access faster. Objects are identified by a pair `<group, key>` which could be mapped, for example, to a disk drive identifier and sector identifier. Read and write operations are performed using externally-supplied functions, the cache is device-agnostic.

The cache uses internally an hash table, the size of the table should be dimensioned to minimize the risk of hash collisions, a factor of two is usually acceptable, it depends on the specific application requirements.

Operations defined for caches:

- **Get Object**: Retrieves an object from cache, if not present then an empty buffer is returned.

- **Read Object**: Retrieves an object from cache, if not present a buffer is allocated and the object is read from the media.

- **Release Object**: Releases an object to the cache handling the media update, if required.

#### Precondition

In order to use the pipes APIs the `CH_CFG_USE_OBJ_CACHES` option must be enabled in `chconf.h`.

#### Note

Compatible with RT and NIL.

### 9.36 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 an hash element header.
- typedef struct ch_oc_lru_header oc_lru_header_t
  Type of an LRU element header.
- typedef struct ch_oc_object oc_object_t
  Type of a cached object.
- typedef struct ch_objects_cache objects_cache_t
  Type of a cache object.
- typedef bool(∗ oc_readf_t) (objects_cache_t ∗ocp, oc_object_t ∗objp, bool async)
  Object read function.
- typedef bool(∗ oc_writef_t) (objects_cache_t ∗ocp, oc_object_t ∗objp, bool async)
  Object write function.

# Functions

- void chCacheObjectInit (objects_cache_t ∗ocp, ucnt_t hashn, oc_hash_header_t ∗hashp, ucnt_t objn, size_t objsz, void ∗objvp, oc_readf_t readf, oc_writef_t writef)
  Initializes a objects_cache_t object.
- oc_object_t ∗chCacheGetObject (objects_cache_t ∗ocp, uint32_t group, uint32_t key)
  Retrieves an object from the cache.
- void 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.
9.36.1 Detailed Description

Objects Caches macros and structures.

9.37 chobjfifos.h File Reference

Objects FIFO structures and macros.

Data Structures

- struct ch_objects_fifo
  Type of an objects FIFO.

Typedefs

- typedef struct ch_objects_fifo objects_fifo_t
  Type of an objects FIFO.

Functions

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

9.37.1 Detailed Description

Objects FIFO structures and macros.

This module implements a generic FIFO queue of objects by coupling a Guarded Memory Pool (for objects storage) and a MailBox.

On the sender side free objects are taken from the pool, filled and then sent to the receiver, on the receiver side objects are fetched, used and then returned to the pool. Operations defined for object FIFOs:

• Take: An object is taken from the pool of the free objects, can be blocking.
• Return: An object is returned to the pool of the free objects, it is guaranteed to be non-blocking.
• Send: An object is sent through the mailbox, it is guaranteed to be non-blocking.
• Receive: An object is received from the mailbox, can be blocking.

9.38 chpipes.c File Reference

Pipes code.

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

Functions

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

Pipes code.

Byte pipes.

Operation mode

A pipe is an asynchronous communication mechanism. Operations defined for mailboxes:

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

Precondition

In order to use the pipes APIs the `CH_CFG_USE_PIPES` option must be enabled in `chconf.h`.

Note

Compatible with RT and NIL.

9.39 chpipes.h File Reference

Pipes macros and structures.

Data Structures

- `struct pipe_t`
  
  *Structure representing a pipe object.*

Macros

- `#define _PIPE_DATA(name, buffer, size)`
  
  *Data part of a static pipe initializer.*

- `#define PIPE_DECL(name, buffer, size) pipe_t name = _PIPE_DATA(name, buffer, size)`
  
  *Static pipe initializer.*
Functions

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

9.39.1 Detailed Description

Pipes macros and structures.

9.40 chregistry.c File Reference

Threads registry code.

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

Functions

- thread_t * chRegFirstThread (void)
  Returns the first thread in the system.
- thread_t * chRegNextThread (thread_t *tp)
  Returns the thread next to the specified one.
- thread_t * chRegFindThreadByName (const char *name)
  Retrieves a thread pointer by name.
- thread_t * chRegFindThreadByPointer (thread_t *tp)
  Confirms that a pointer is a valid thread pointer.
- thread_t * chRegFindThreadByWorkingArea (stkalign_t *wa)
  Confirms that a working area is being used by some active thread.

9.40.1 Detailed Description

Threads registry code.
9.41  chregistry.h File Reference

Threads registry macros and structures.

Data Structures

- struct chdebug_t
  
  ChibiOS/RT memory signature record.

Macros

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

Functions

- thread_t * chRegFirstThread (void)
  
  Returns the first thread in the system.
- thread_t * chRegNextThread (thread_t *tp)
  
  Returns the thread next to the specified one.
- thread_t * chRegFindThreadByName (const char *name)
  
  Retrieves a thread pointer by name.
- thread_t * chRegFindThreadByPointer (thread_t *tp)
  
  Confirms that a pointer is a valid thread pointer.
- thread_t * chRegFindThreadByWorkingArea (stkalign_t *wa)
  
  Confirms that a working area is being used by some active thread.
- static void chRegSetThreadName (const char *name)
  
  Sets the current thread name.
- static const char * chRegGetThreadNameX (thread_t *tp)
  
  Returns the name of the specified thread.
- static void chRegSetThreadNameX (thread_t *tp, const char *name)
  
  Changes the name of the specified thread.

9.41.1  Detailed Description

Threads registry macros and structures.

9.42  chrestrictions.h File Reference

Licensing restrictions header.
9.42.1 Detailed Description

Licensing restrictions header.

9.43 chschd.c File Reference

Scheduler code.

#include "ch.h"

Functions

- void ch_sch_prio_insert (ch_queue_t *tp, ch_queue_t *qp)
  
  Inserts a thread into a priority ordered queue.
- void _scheduler_init (void)
  
  Scheduler initialization.
- thread_t * chSchReadyI (thread_t *tp)
  
  Inserts a thread in the Ready List placing it behind its peers.
- thread_t * chSchReadyAheadI (thread_t *tp)
  
  Inserts a thread in the Ready List placing it ahead its peers.
- void chSchGoSleepS (tstate_t newstate)
  
  Puts the current thread to sleep into the specified state.
- msg_t chSchGoSleepTimeoutS (tstate_t newstate, sysinterval_t timeout)
  
  Puts the current thread to sleep into the specified state with timeout specification.
- void chSchWakeupS (thread_t *ntp, msg_t msg)
  
  Wakes up a thread.
- void chSchRescheduleS (void)
  
  Performs a reschedule if a higher priority thread is runnable.
- bool chSchIsPreemptionRequired (void)
  
  Evaluates if preemption is required.
- void chSchDoRescheduleBehind (void)
  
  Switches to the first thread on the runnable queue.
- void chSchDoRescheduleAhead (void)
  
  Switches to the first thread on the runnable queue.
- void chSchDoReschedule (void)
  
  Switches to the first thread on the runnable queue.

Variables

- ch_system_t ch
  
  System data structures.

9.43.1 Detailed Description

Scheduler code.
9.44 chschd.h File Reference

Scheduler macros and structures.

Data Structures

- struct ch_threads_queue
  Structure representing a threads queue.
- struct ch_thread
  Structure representing a thread.
- struct ch_delta_list
  Virtual Timer delta list element and header structure.
- struct ch_virtual_timer
  Structure representing a Virtual Timer.
- struct ch_virtual_timers_list
  Structure representing a virtual timers list header.
- struct ch_system_debug
  System debug data structure.
- struct ch_system
  System data structure.

Macros

- #define firstprio(rlp) ((rlp)->next->prio)
  Returns the priority of the first thread on the given ready list.
- #define currp ch.rlist.current
  Current thread pointer access macro.

Wakeup status codes

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

Priority constants

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

- `#define HIGHPRIO (tprio_t)255`
  
  Highest priority.

### Thread states

- `#define CH_STATE_READY (tstate_t)0`
  
  Waiting on the ready list.

- `#define CH_STATE_CURRENT (tstate_t)1`
  
  Currently running.

- `#define CH_STATE_WTSTART (tstate_t)2`
  
  Just created.

- `#define CH_STATE_SUSPENDED (tstate_t)3`
  
  Suspended state.

- `#define CH_STATE_QUEUE (tstate_t)4`
  
  On a queue.

- `#define CH_STATE_WTSEM (tstate_t)5`
  
  On a semaphore.

- `#define CH_STATE_WTMTX (tstate_t)6`
  
  On a mutex.

- `#define CH_STATE_WTCND (tstate_t)7`
  
  On a cond.variable.

- `#define CH_STATE_SLEEPING (tstate_t)8`
  
  Sleeping.

- `#define CH_STATE_WTEXIT (tstate_t)9`
  
  Waiting a thread.

- `#define CH_STATE_WTOREVT (tstate_t)10`
  
  One event.

- `#define CH_STATE_WTANDEVT (tstate_t)11`
  
  Several events.

- `#define CH_STATE_SNDMSGQ (tstate_t)12`
  
  Sending a message, in queue.

- `#define CH_STATE_SNDMSG (tstate_t)13`
  
  Sent a message, waiting answer.

- `#define CH_STATE_WTMSG (tstate_t)14`
  
  Waiting for a message.

- `#define CH_STATE_FINAL (tstate_t)15`
  
  Thread terminated.

- `#define CH_STATE_NAMES`
  
  Thread states as array of strings.

### Thread flags and attributes

- `#define CH_FLAG_MODE_MASK (tmode_t)3U`
Thread memory mode mask.

- `#define CH_FLAG_MODE_STATIC (tmode_t)0U`
  Static thread.

- `#define CH_FLAG_MODE_HEAP (tmode_t)1U`
  Thread allocated from a Memory Heap.

- `#define CH_FLAG_MODE_MPOOL (tmode_t)2U`
  Thread allocated from a Memory Pool.

- `#define CH_FLAG_TERMINATE (tmode_t)4U`
  Termination requested flag.

**Typedefs**

- `typedef struct ch_delta_list delta_list_t`
  Type of a Virtual Timer structure.

**Functions**

- `void _scheduler_init (void)`
  Scheduler initialization.

- `thread_t * chSchReadyI (thread_t *tp)`
  Inserts a thread in the Ready List placing it behind its peers.

- `thread_t * chSchReadyAheadI (thread_t *tp)`
  Inserts a thread in the Ready List placing it ahead its peers.

- `void chSchGoSleepS (tstate_t newstate)`
  Puts the current thread to sleep into the specified state.

- `msg_t chSchGoSleepTimeoutS (tstate_t newstate, sysinterval_t timeout)`
  Puts the current thread to sleep into the specified state with timeout specification.

- `void chSchWakeupS (thread_t *ntp, msg_t msg)`
  Wakes up a thread.

- `void chSchRescheduleS (void)`
  Performs a reschedule if a higher priority thread is runnable.

- `bool chSchIsPreemptionRequired (void)`
  Evaluates if preemption is required.

- `void chSchIsRescRequiredI (void)`
  Determines if the current thread must reschedule.

- `static bool chSchCanYieldS (void)`
  Determines if yielding is possible.

- `static void chSchDoYieldS (void)`
  Yields the time slot.

- `static void chSchPreemption (void)`
  Inline-able preemption code.
9.44.1 Detailed Description

Scheduler macros and structures.

9.45 chsem.c File Reference

Semaphore code.

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

**Functions**

- `void chSemObjectInit (semaphore_t *sp, cnt_t n)`
  - Initializes a semaphore with the specified counter value.
- `void chSemResetWithMessage (semaphore_t *sp, cnt_t n, msg_t msg)`
  - Performs a reset operation on the semaphore.
- `void chSemResetWithMessageI (semaphore_t *sp, cnt_t n, msg_t msg)`
  - Performs a reset operation on the semaphore.
- `msg_t chSemWait (semaphore_t *sp)`
  - Performs a wait operation on a semaphore.
- `msg_t chSemWaitS (semaphore_t *sp)`
  - Performs a wait operation on a semaphore.
- `msg_t chSemWaitTimeout (semaphore_t *sp, sysinterval_t timeout)`
  - Performs a wait operation on a semaphore with timeout specification.
- `msg_t chSemWaitTimeoutS (semaphore_t *sp, sysinterval_t timeout)`
  - Performs a wait operation on a semaphore with timeout specification.
- `void chSemSignal (semaphore_t *sp)`
  - Performs a signal operation on a semaphore.
- `void chSemSignalI (semaphore_t *sp)`
  - Performs a signal operation on a semaphore.
- `void chSemAddCounterI (semaphore_t *sp, cnt_t n)`
  - Adds the specified value to the semaphore counter.
- `msg_t chSemSignalWait (semaphore_t *sps, semaphore_t *spw)`
  - Performs atomic signal and wait operations on two semaphores.

9.45.1 Detailed Description

Semaphore code.

9.46 chsem.h File Reference

Semaphore macros and structures.
Data Structures

- struct ch_semaphore
  Semaphore structure.

Macros

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

Typedefs

- typedef struct ch_semaphore semaphore_t
  Semaphore structure.

Functions

- void chSemObjectInit (semaphore_t *sp, cnt_t n)
  Initializes a semaphore with the specified counter value.
- void chSemResetWithMessage (semaphore_t *sp, cnt_t n, msg_t msg)
  Performs a reset operation on the semaphore.
- void chSemResetWithMessageI (semaphore_t *sp, cnt_t n, msg_t msg)
  Performs a reset operation on the semaphore.
- msg_t chSemWait (semaphore_t *sp)
  Performs a wait operation on a semaphore.
- msg_t chSemWaitS (semaphore_t *sp)
  Performs a wait operation on a semaphore.
- msg_t chSemWaitTimeout (semaphore_t *sp, sysinterval_t timeout)
  Performs a wait operation on a semaphore with timeout specification.
- msg_t chSemWaitTimeoutS (semaphore_t *sp, sysinterval_t timeout)
  Performs a wait operation on a semaphore with timeout specification.
- void chSemSignal (semaphore_t *sp)
  Performs a signal operation on a semaphore.
- void chSemSignalI (semaphore_t *sp)
  Performs a signal operation on a semaphore.
- void chSemAddCounterI (semaphore_t *sp, cnt_t n)
  Adds the specified value to the semaphore counter.
- msg_t chSemSignalWait (semaphore_t *sps, semaphore_t *spw)
  Performs atomic signal and wait operations on two semaphores.
- static void chSemReset (semaphore_t *sp, cnt_t n)
  Performs a reset operation on the semaphore.
- static void chSemResetI (semaphore_t *sp, cnt_t n)
  Performs a reset operation on the semaphore.
- static void chSemFastWaitI (semaphore_t *sp)
  Decreases the semaphore counter.
- static void chSemFastSignall (semaphore_t *sp)
  Increases the semaphore counter.
- static cnt_t chSemGetCounterI (const semaphore_t *sp)
  Returns the semaphore counter current value.
9.46.1 Detailed Description

Semaphores macros and structures.

9.47 chstats.c File Reference

Statistics module code.

#include "ch.h"

Functions

- void _stats_init (void)
  Initializes the statistics module.
- void _stats_increase_irq (void)
  Increases the IRQ counter.
- void _stats_ctxswc (thread_t *ntp, thread_t *otp)
  Updates context switch related statistics.
- void _stats_start_measure_crit_thd (void)
  Starts the measurement of a thread critical zone.
- void _stats_stop_measure_crit_thd (void)
  Stops the measurement of a thread critical zone.
- void _stats_start_measure_crit_isr (void)
  Starts the measurement of an ISR critical zone.
- void _stats_stop_measure_crit_isr (void)
  Stops the measurement of an ISR critical zone.

9.47.1 Detailed Description

Statistics module code.

9.48 chstats.h File Reference

Statistics module macros and structures.

Data Structures

- struct kernel_stats_t
  Type of a kernel statistics structure.
Functions

- void _stats_init (void)
  Initializes the statistics module.
- void _stats_increase_irq (void)
  Increases the IRQ counter.
- void _stats_ctxswc (thread_t *ntp, thread_t *otp)
  Updates context switch related statistics.
- void _stats_start_measure_crit_thd (void)
  Starts the measurement of a thread critical zone.
- void _stats_stop_measure_crit_thd (void)
  Stops the measurement of a thread critical zone.
- void _stats_start_measure_crit_isr (void)
  Starts the measurement of an ISR critical zone.
- void _stats_stop_measure_crit_isr (void)
  Stops the measurement of an ISR critical zone.

9.48.1 Detailed Description

Statistics module macros and structures.

9.49 chsys.c File Reference

System related code.

#include "ch.h"

Functions

- THD_WORKING_AREA (ch_idle_thread_wa, PORT_IDLE_THREAD_STACK_SIZE)
  Idle thread working area.
- static void _idle_thread (void *)
  This function implements the idle thread infinite loop.
- void chSysInit (void)
  ChibiOS/RT initialization.
- void chSysHalt (const char *)
  Halts the system.
- bool chSysIntegrityCheckI (unsigned testmask)
  System integrity check.
- void chSysTimerHandlerI (void)
  Handles time ticks for round robin preemption and timer increments.
- syssts_t chSysGetStatusAndLockX (void)
  Returns the execution status and enters a critical zone.
- void chSysRestoreStatusX (syssts_t sts)
  Restores the specified execution status and leaves a critical zone.
- bool chSysIsCounterWithinX (rtcnt_t cnt, rtcnt_t start, rtcnt_t end)
  Realtime window test.
- void chSysPolledDelayX (rtcnt_t cycles)
  Polled delay.
9.49.1 Detailed Description

System related code.

9.50 chsys.h File Reference

System related macros and structures.

Macros

* #define chSysGetRealtimeCounterX() (rtcnt_t)port_rt_get_counter_value()
  Returns the current value of the system real time counter.

* #define chSysSwitch(ntp, otp)
  Performs a context switch.

Masks of executable integrity checks.

* #define CH_INTEGRITY_RLIST 1U
* #define CH_INTEGRITY_VTLIST 2U
* #define CH_INTEGRITY_REGISTRY 4U
* #define CH_INTEGRITY_PORT 8U

ISRs abstraction macros

* #define CH_IRQ_IS_VALID_PRIORITY(prio) PORT_IRQ_IS_VALID_PRIORITY(prio)
  Priority level validation macro.

* #define CH_IRQ_IS_VALID_KERNEL_PRIORITY(prio) PORT_IRQ_IS_VALID_KERNEL_PRIORITY(prio)
  Priority level validation macro.

* #define CH_IRQ_PROLOGUE()
  IRQ handler enter code.

* #define CH_IRQ_EPILOGUE()
  IRQ handler exit code.

* #define CH_IRQ_HANDLER(id) PORT_IRQ_HANDLER(id)
  Standard normal IRQ handler declaration.

Fast ISRs abstraction macros

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

Time conversion utilities for the realtime counter

* #define S2RTC(freq, sec) (((freq) * (sec)))
  Seconds to realtime counter.

* #define MS2RTC(freq, msec) (rtcnt_t)(((freq) + 999UL) / 1000UL) * (msec)
  Milliseconds to realtime counter.

* #define US2RTC(freq, usec) (rtcnt_t)(((freq) + 999999UL) / 1000000UL) * (usec)
  Microseconds to realtime counter.

* #define RTC2S(freq, n) (((n) - 1UL) / (freq)) + 1UL
  Realtime counter cycles to seconds.

* #define RTC2MS(freq, n) (((n) - 1UL) / (freq) / 1000UL)) + 1UL
  Realtime counter cycles to milliseconds.

* #define RTC2US(freq, n) (((n) - 1UL) / (freq) / 1000000UL)) + 1UL
  Realtime counter cycles to microseconds.
Functions

- void chSysInit (void)
  ChibiOS/RT initialization.
- bool chSysIntegrityCheckI (unsigned testmask)
  System integrity check.
- void chSysTimerHandlerI (void)
  Handles time ticks for round robin preemption and timer increments.
- syssts_t chSysGetStatusAndLockX (void)
  Returns the execution status and enters a critical zone.
- void chSysRestoreStatusX (syssts_t sts)
  Restores the specified execution status and leaves a critical zone.
- bool chSysIsCounterWithinX (rtcnt_t cnt, rtcnt_t start, rtcnt_t end)
  Realtime window test.
- void chSysPolledDelayX (rtcnt_t cycles)
  Polled delay.
- static void chSysDisable (void)
  Raises the system interrupt priority mask to the maximum level.
- static void chSysSuspend (void)
  Raises the system interrupt priority mask to system level.
- static void chSysEnable (void)
  Lowers the system interrupt priority mask to user level.
- static void chSysLock (void)
  Enters the kernel lock state.
- static void chSysUnlock (void)
  Leaves the kernel lock state.
- static void chSysLockFromISR (void)
  Enters the kernel lock state from within an interrupt handler.
- static void chSysUnlockFromISR (void)
  Leaves the kernel lock state from within an interrupt handler.
- static void chSysUnconditionalLock (void)
  Unconditionally enters the kernel lock state.
- static void chSysUnconditionalUnlock (void)
  Unconditionally leaves the kernel lock state.
- static thread_t ∗ chSysGetIdleThreadX (void)
  Returns a pointer to the idle thread.

9.50.1 Detailed Description

System related macros and structures.

9.51 chsystypes.h File Reference

System types header.

Macros

- #define __CH_STRINGIFY(a) #a
  Utility to make the parameter a quoted string.
**Typedefs**

- `typedef struct ch_thread thread_t`
  *Type of a thread structure.*
- `typedef thread_t * thread_reference_t`
  *Type of a thread reference.*
- `typedef struct ch_threads_list threads_list_t`
  *Type of a generic threads single link list, it works like a stack.*
- `typedef struct ch_threads_queue threads_queue_t`
  *Type of a generic threads bidirectional linked list header and element.*
- `typedef struct ch_ready_list ready_list_t`
  *Type of a ready list header.*
- `typedef void(* vtfunc_t) (void *)`
  *Type of a Virtual Timer callback function.*
- `typedef struct ch_virtual_timer virtual_timer_t`
  *Type of a Virtual Timer structure.*
- `typedef struct ch_virtual_timers_list virtual_timers_list_t`
  *Type of virtual timers list header.*
- `typedef struct ch_system_debug system_debug_t`
  *Type of a system debug structure.*
- `typedef struct ch_system ch_system_t`
  *Type of system data structure.*

9.51.1 Detailed Description

System types header.

9.52 chthreads.c File Reference

Threads code.

`#include "ch.h"`

**Functions**

- `thread_t * _thread_init (thread_t *tp, const char *name, tprio_t prio)`  
  Initializes a thread structure.
- `void _thread_memfill (uint8_t *startp, uint8_t *endp, uint8_t v)`  
  Memory fill utility.
- `thread_t * chThdCreateSuspendedI (const thread_descriptor_t *tdp)`  
  Creates a new thread into a static memory area.
- `thread_t * chThdCreateSuspended (const thread_descriptor_t *tdp)`  
  Creates a new thread into a static memory area.
- `thread_t * chThdCreateI (const thread_descriptor_t *tdp)`  
  Creates a new thread into a static memory area.
- `thread_t * chThdCreate (const thread_descriptor_t *tdp)`  
  Creates a new thread into a static memory area.`
9.52 chthreads.c File Reference

- thread_t * chThdCreateStatic (void *wsp, size_t size, tprio_t prio, tfunc_t pf, void *arg)
  Creates a new thread into a static memory area.

- thread_t * chThdStart (thread_t *tp)
  Resumes a thread created with chThdCreateI().

- thread_t * chThdAddRef (thread_t *tp)
  Adds a reference to a thread object.

- void chThdRelease (thread_t *tp)
  Releases a reference to a thread object.

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

- void chThdExitS (msg_t msg)
  Terminates the current thread.

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

- tprio_t chThdSetPriority (tprio_t newprio)
  Changes the running thread priority level then reschedules if necessary.

- void chThdTerminate (thread_t *tp)
  Requests a thread termination.

- void chThdSleep (sysinterval_t time)
  Suspends the invoking thread for the specified time.

- void chThdSleepUntil (systime_t time)
  Suspends the invoking thread until the system time arrives to the specified value.

- systime_t chThdSleepUntilWindowed (systime_t prev, systime_t next)
 Suspends the invoking thread until the system time arrives to the specified value.

- void chThdYield (void)
  Yields the time slot.

- msg_t chThdSuspendS (thread_reference_t *trp)
  Sends the current thread sleeping and sets a reference variable.

- msg_t chThdSuspendTimeoutS (thread_reference_t *trp, sysinterval_t timeout)
  Sends the current thread sleeping and sets a reference variable.

- void chThdResumeI (thread_reference_t *trp, msg_t msg)
  Wakes up a thread waiting on a thread reference object.

- void chThdResumeS (thread_reference_t *trp, msg_t msg)
  Wakes up a thread waiting on a thread reference object.

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

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

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

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

9.52.1 Detailed Description

Threads code.
9.53  chthreads.h File Reference

Threads module macros and structures.

Data Structures

• struct thread_descriptor_t
  Type of a thread descriptor.

Macros

Threads queues

• #define _THREADS_QUEUE_DATA(name) {_CH_QUEUE_DATA(name)}
  Data part of a static threads queue object initializer.
• #define THREADS_QUEUE_DECL(name) threads_queue_t name = _THREADS_QUEUE_DATA(name)
  Static threads queue object initializer.

Working Areas

• #define THD_WORKING_AREA_SIZE(n) MEM_ALIGN_NEXT(sizeof(thread_t) + PORT_WA_SIZE(n), PORT_STACK_ALIGN)
  Calculates the total Working Area size.
• #define THD_WORKING_AREA(s, n) PORT_WORKING_AREA(s, n)
  Static working area allocation.
• #define THD_WORKING_AREA_BASE(s) ((stkalign_t*)(s))
  Base of a working area casted to the correct type.
• #define THD_WORKING_AREA_END(s)
  End of a working area casted to the correct type.

Threads abstraction macros

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

Macro Functions

• #define chThdSleepSeconds(sec) chThdSleep(TIME_S2I(sec))
  Delays the invoking thread for the specified number of seconds.
• #define chThdSleepMilliseconds(msec) chThdSleep(TIME_MS2I(msec))
  Delays the invoking thread for the specified number of milliseconds.
• #define chThdSleepMicroseconds(usec) chThdSleep(TIME_US2I(usec))
  Delays the invoking thread for the specified number of microseconds.

Typedefs

• typedef void(*tfunc_t)(void p)
  Thread function.
### Functions

- **thread_t * _thread_init (thread_t *tp, const char *name, tprio_t prio)**  
  Initializes a thread structure.

- **void _thread_memfill (uint8_t *startp, uint8_t *endp, uint8_t v)**  
  Memory fill utility.

- **thread_t * chThdCreateSuspended1 (const thread_descriptor_t *tdp)**  
  Creates a new thread into a static memory area.

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

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

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

- **thread_t * chThdCreateStatic (void *wsp, size_t size, tprio_t prio, tfunc_t pf, void *arg)**  
  Creates a new thread into a static memory area.

- **thread_t * chThdStart (thread_t *tp)**  
  Resumes a thread created with `chThdCreateI()`.

- **thread_t * chThdAddRef (thread_t *tp)**  
  Adds a reference to a thread object.

- **void chThdRelease (thread_t *tp)**  
  Releases a reference to a thread object.

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

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

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

- **tprio_t chThdSetPriority (tprio_t newprio)**  
  Changes the running thread priority level then reschedules if necessary.

- **void chThdTerminate (thread_t *tp)**  
  Requests a thread termination.

- **msg_t chThdSuspendS (thread_reference_t *trp)**  
  Sends the current thread sleeping and sets a reference variable.

- **msg_t chThdSuspendTimeoutS (thread_reference_t *trp, sysinterval_t timeout)**  
  Sends the current thread sleeping and sets a reference variable.

- **void chThdResumeI (thread_reference_t *trp, msg_t msg)**  
  Wakes up a thread waiting on a thread reference object.

- **void chThdResumeS (thread_reference_t *trp, msg_t msg)**  
  Wakes up a thread waiting on a thread reference object.

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

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

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

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

- **void chThdSleep (sysinterval_t time)**  
  Suspends the invoking thread for the specified time.

- **void chThdSleepUntil (systime_t time)**  
  Suspends the invoking thread until the specified time.
Suspend the invoking thread until the system time arrives to the specified value.

- **systime_t chThdSleepUntilWindowed (systime_t prev, systime_t next)**
  
  Suspend the invoking thread until the system time arrives to the specified value.

- **void chThdYield (void)**
  
  Yields the time slot.

- **static thread_t * chThdGetSelfX (void)**
  
  Returns a pointer to the current thread_t.

- **static tprio_t chThdGetPriorityX (void)**
  
  Returns the current thread priority.

- **static systime_t chThdGetTicksX (thread_t *tp)**
  
  Returns the number of ticks consumed by the specified thread.

- **static stkalign_t * chThdGetWorkingAreaX (thread_t *tp)**
  
  Returns the working area base of the specified thread.

- **static bool chThdTerminatedX (thread_t *tp)**
  
  Verifies if the specified thread is in the CH_STATE_FINAL state.

- **static bool chThdShouldTerminateX (void)**
  
  Verifies if the current thread has a termination request pending.

- **static thread_t * chThdStartI (thread_t *tp)**
  
  Resumes a thread created with chThdCreateI().

- **static void chThdSleepS (sysinterval_t ticks)**
  
  Suspend the invoking thread for the specified number of ticks.

- **static void chThdQueueObjectInit (threads_queue_t *tqp)**
  
  Initializes a threads queue object.

- **static bool chThdQueueIsEmptyI (threads_queue_t *tqp)**
  
  Evaluates to true if the specified queue is empty.

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

### 9.53.1 Detailed Description

Threads module macros and structures.

### 9.54 chtime.h File Reference

Time and intervals macros and structures.

**Macros**

**Special time constants**

- `#define TIME_IMMEDIATE ((sysinterval_t)0)`
  
  Zero interval specification for some functions with a timeout specification.

- `#define TIME_INFINITE ((sysinterval_t)-1)`
  
  Infinite interval specification for all functions with a timeout specification.

- `#define TIME_MAX_INTERVAL ((sysinterval_t)-2)`
  
  Maximum interval constant usable as timeout.

- `#define TIME_MAX_SYSTIME ((systime_t)-1)`
  
  Maximum system of system time before it wraps.
Fast time conversion utilities

• #define TIME_S2I(secs) ((sysinterval_t)((time_conv_t)(secs) \* (time_conv_t)CH_CFG_ST_FREQUENCY))
  Seconds to time interval.
• #define TIME_MS2I(msec)
  Milliseconds to time interval.
• #define TIME_US2I(usec)
  Microseconds to time interval.
• #define TIME_I2S(interval)
  Time interval to seconds.
• #define TIME_I2MS(interval)
  Time interval to milliseconds.
• #define TIME_I2US(interval)
  Time interval to microseconds.

Typedefs

• typedef uint64_t systime_t
  Type of system time.
• typedef uint64_t sysinterval_t
  Type of time interval.
• typedef uint32_t time_secs_t
  Type of seconds.
• typedef uint32_t time_msecs_t
  Type of milliseconds.
• typedef uint32_t time_usecs_t
  Type of microseconds.
• typedef uint64_t time_conv_t
  Type of time conversion variable.

Functions

Secure time conversion utilities

• static sysinterval_t chTimeS2I (time_secs_t secs)
  Seconds to time interval.
• static sysinterval_t chTimeMS2I (time_msecs_t msec)
  Milliseconds to time interval.
• static sysinterval_t chTimeUS2I (time_usecs_t usec)
  Microseconds to time interval.
• static time_secs_t chTimeI2S (sysinterval_t interval)
  Time interval to seconds.
• static time_msecs_t chTimeI2MS (sysinterval_t interval)
  Time interval to milliseconds.
• static time_usecs_t chTimeI2US (sysinterval_t interval)
  Time interval to microseconds.
• static systime_t chTimeAddX (systime_t systime, sysinterval_t interval)
  Adds an interval to a system time returning a system time.
• static sysinterval_t chTimeDiffX (systime_t start, systime_t end)
  Subtracts two system times returning an interval.
• static bool chTimelsInRangeX (systime_t time, systime_t start, systime_t end)
  Checks if the specified time is within the specified time range.
9.54.1 Detailed Description

Time and intervals macros and structures.

9.55 chtm.c File Reference

Time Measurement module code.

#include "ch.h"

Macros

• #define TM_CALIBRATION_LOOP 4U
  Number of iterations in the calibration loop.

Functions

• void _tm_init (void)
  Initializes the time measurement unit.
• void chTMObjectInit (time_measurement_t *tmp)
  Initializes a TimeMeasurement object.
• NOINLINE void chTMStartMeasurementX (time_measurement_t *tmp)
  Starts a measurement.
• NOINLINE void chTMStopMeasurementX (time_measurement_t *tmp)
  Stops a measurement.
• NOINLINE void chTMChainMeasurementToX (time_measurement_t *tmp1, time_measurement_t *tmp2)
  Stops a measurement and chains to the next one using the same time stamp.

9.55.1 Detailed Description

Time Measurement module code.

9.56 chtm.h File Reference

Time Measurement module macros and structures.

Data Structures

• struct tm_calibration_t
  Type of a time measurement calibration data.
• struct time_measurement_t
  Type of a Time Measurement object.
Functions

- void _tm_init (void)
  Initializes the time measurement unit.
- void chTMObjectInit (time_measurement_t *tmp)
  Initializes a TimeMeasurement object.
- NOINLINE void chTMStartMeasurementX (time_measurement_t *tmp)
  Starts a measurement.
- NOINLINE void chTMStopMeasurementX (time_measurement_t *tmp)
  Stops a measurement.
- NOINLINE void chTMChainMeasurementToX (time_measurement_t *tmp1, time_measurement_t *tmp2)
  Stops a measurement and chains to the next one using the same time stamp.

9.56.1 Detailed Description

Time Measurement module macros and structures.

9.57 chtrace.c File Reference

Tracer code.

#include "ch.h"

Functions

- static NOINLINE void trace_next (void)
  Writes a time stamp and increases the trace buffer pointer.
- void _trace_init (void)
  Trace circular buffer subsystem initialization.
- void _trace_switch (thread_t *ntp, thread_t *otp)
  Inserts in the circular debug trace buffer a context switch record.
- void _trace_isr_enter (const char *isr)
  Inserts in the circular debug trace buffer an ISR-enter record.
- void _trace_isr_leave (const char *isr)
  Inserts in the circular debug trace buffer an ISR-leave record.
- void _trace_halt (const char *reason)
  Inserts in the circular debug trace buffer an halt record.
- void chDbgWriteTrace1 (void *up1, void *up2)
  Adds an user trace record to the trace buffer.
- void chDbgWriteTrace (void *up1, void *up2)
  Adds an user trace record to the trace buffer.
- void chDbgSuspendTrace1 (uint16_t mask)
  Suspends one or more trace events.
- void chDbgSuspendTrace (uint16_t mask)
  Suspends one or more trace events.
- void chDbgResumeTrace1 (uint16_t mask)
  Resumes one or more trace events.
- void chDbgResumeTrace (uint16_t mask)
  Resumes one or more trace events.
9.57.1 Detailed Description

Tracer code.

9.58 chtrace.h File Reference

Tracer macros and structures.

Data Structures

- struct ch_trace_event_t
  Trace buffer record.
- struct ch_trace_buffer_t
  Trace buffer header.

Macros

Trace record types

- #define CH_TRACE_TYPE_UNUSED 0U
- #define CH_TRACE_TYPE_SWITCH 1U
- #define CH_TRACE_TYPE_ISR_ENTER 2U
- #define CH_TRACE_TYPE_ISR_LEAVE 3U
- #define CH_TRACE_TYPE_HALT 4U
- #define CH_TRACE_TYPE_USER 5U

Events to trace

- #define CH_DBG_TRACE_MASK_DISABLED 255U
- #define CH_DBG_TRACE_MASK_NONE 0U
- #define CH_DBG_TRACE_MASK_SWITCH 1U
- #define CH_DBG_TRACE_MASK_ISR 2U
- #define CH_DBG_TRACE_MASK_HALT 4U
- #define CH_DBG_TRACE_MASK_USER 8U
- #define CH_DBG_TRACE_MASK_SLOW
- #define CH_DBG_TRACE_MASK_ALL

Debug related settings

- #define CH_DBG_TRACE_MASK CH_DBG_TRACE_MASK_DISABLED
  Trace buffer entries.
- #define CH_DBG_TRACE_BUFFER_SIZE 128
  Trace buffer entries.
Functions

- **void _trace_init (void)**
  
  Trace circular buffer subsystem initialization.

- **void _trace_switch (thread_t *ntp, thread_t *otp)**

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

- **void _trace_isr_enter (const char *isr)**

  Inserts in the circular debug trace buffer an ISR-enter record.

- **void _trace_isr_leave (const char *isr)**

  Inserts in the circular debug trace buffer an ISR-leave record.

- **void _trace_halt (const char *reason)**

  Inserts in the circular debug trace buffer an halt record.

- **void chDbgWriteTraceI (void *up1, void *up2)**

  Adds an user trace record to the trace buffer.

- **void chDbgWriteTrace (void *up1, void *up2)**

  Adds an user trace record to the trace buffer.

- **void chDbgSuspendTraceI (uint16_t mask)**

  Suspends one or more trace events.

- **void chDbgSuspendTrace (uint16_t mask)**

  Suspends one or more trace events.

- **void chDbgResumeTraceI (uint16_t mask)**

  Resumes one or more trace events.

- **void chDbgResumeTrace (uint16_t mask)**

  Resumes one or more trace events.

9.58.1 Detailed Description

Tracer macros and structures.

9.59 chversion.h File Reference

Version Module macros and structures.

Macros

- **#define _CHIBIOS_**

  ChibiOS product identification macro.

- **#define CH_VERSION_STABLE 1**

  Stable release flag.

ChibiOS version identification

- **#define CH_VERSION "20.3.4"**

  ChibiOS version string.

- **#define CH_VERSION_YEAR 20**

  ChibiOS version release year.

- **#define CH_VERSION_MONTH 3**

  ChibiOS version release month.

- **#define CH_VERSION_PATCH 4**

  ChibiOS version patch number.

- **#define CH_VERSION_NICKNAME "Conca"**

  ChibiOS version nickname.
9.59.1 Detailed Description

Version Module macros and structures.

9.60 chvt.c File Reference

Time and Virtual Timers module code.

#include "ch.h"

Functions

- static void vt_init (delta_list_t *dlhp)
  
  Delta list initialization.
- static bool vt_is_empty (delta_list_t *dlhp)
  
  List empty check.
- static bool vt_is_last (delta_list_t *dlhp, delta_list_t *dlp)
  
  Last timer in the list check.
- static bool vt_is_first (delta_list_t *dlhp, delta_list_t *dlp)
  
  First timer in the list check.
- static bool vt_is_timer (delta_list_t *dlhp, delta_list_t *dlp)
  
  Timer check.
- static void vt_insert_after (delta_list_t *dlhp, delta_list_t *dlp, sysinterval_t delta)
  
  Inserts an element after another header element.
- static void vt_insert_before (delta_list_t *dlhp, delta_list_t *dlp, sysinterval_t delta)
  
  Inserts an element before another header element.
- static void vt_insert (delta_list_t *dlhp, delta_list_t *dlep, sysinterval_t delta)
  
  Inserts an element in a delta list.
- static delta_list_t * vt_remove_first (delta_list_t *dlhp)
  
  Dequeues an element from the delta list.
- static delta_list_t * vt_dequeue (delta_list_t *dlp)
  
  Dequeues an element from the delta list.
- static void vt_set_alarm (systime_t now, sysinterval_t delay)
  
  Alarm time setup.
- static void vt_insert_first (virtual_timers_list_t *vtlp, virtual_timer_t *vtp, systime_t now, sysinterval_t delay)
  
  Inserts a timer as first element in a delta list.
- static void vt_enqueue (virtual_timers_list_t *vtlp, virtual_timer_t *vtp, sysinterval_t delay)
  
  Enqueues a timer in a virtual timers list.
- void _vt_init (void)
  
  Virtual Timers initialization.
- void chVTDoSetI (virtual_timer_t *vtp, sysinterval_t delay, vtfunc_t vtfunc, void *par)
  
  Enables a one-shot virtual timer.
- void chVTDoResetI (virtual_timer_t *vtp)
  
  Disables a Virtual Timer.
- sysinterval_t chVTGetRemainingInterval (virtual_timer_t *vtp)
  
  Returns the remaining time interval before next timer trigger.
- void chVTDoTickI (void)
  
  Virtual timers ticker.
9.61.1 Detailed Description

Time and Virtual Timers module code.

9.61 chvt.h File Reference

Time and Virtual Timers module macros and structures.

Functions

- void _vt_init (void)
  
  Virtual Timers initialization.

- void chVTDoSetI (virtual_timer_t *vtp, sysinterval_t delay, vtfunc_t vtfunc, void *par)
  
  Enables a one-shot virtual timer.

- void chVTDoResetI (virtual_timer_t *vtp)
  
  Disables a Virtual Timer.

- void chVTDoTickI (void)
  
  Virtual timers ticker.

- static void chVTObjectInit (virtual_timer_t *vtp)
  
  Initializes a virtual_timer_t object.

- static systime_t chVTGetSystemTimeX (void)
  
  Current system time.

- static systime_t chVTGetSystemTime (void)
  
  Current system time.

- static sysinterval_t chVTTimeElapsedSinceX (systime_t start)
  
  Returns the elapsed time since the specified start time.

- static bool chVTIsSystemTimeWithinX (systime_t start, systime_t end)
  
  Checks if the current system time is within the specified time window.

- static bool chVTIsSystemTimeWithin (systime_t start, systime_t end)
  
  Checks if the current system time is within the specified time window.

- static bool chVTGetTimersStateI (sysinterval_t *timep)
  
  Returns the time interval until the next timer event.

- static bool chVTIsArmedI (const virtual_timer_t *vtp)
  
  Returns true if the specified timer is armed.

- static bool chVTIsArmed (const virtual_timer_t *vtp)
  
  Returns true if the specified timer is armed.

- static void chVTRestsetI (virtual_timer_t *vtp, sysinterval_t delay, vtfunc_t vtfunc, void *par)
  
  Enables a virtual timer.

- static void chVTResetI (virtual_timer_t *vtp)
  
  Disables a Virtual Timer.

- static void chVTReset (virtual_timer_t *vtp)
  
  Disables a Virtual Timer.

- static void chVTSelI (virtual_timer_t *vtp, sysinterval_t delay, vtfunc_t vtfunc, void *par)
  
  Enables a virtual timer.

- static void chVTSel (virtual_timer_t *vtp, sysinterval_t delay, vtfunc_t vtfunc, void *par)
  
  Enables a virtual timer.

9.61.1 Detailed Description

Time and Virtual Timers module macros and structures.
Index

_BSEMAPHORE_DATA
  Binary Semaphores, 317
_CHIBIOS_
  Release Information, 20
_CHIBIOS_OSLIB_
  Version Numbers and Identification, 313
_CHIBIOS_RT_
  Version Numbers and Identification, 27
_CH_QUEUE_DATA
  Lists and Queues, 82
_CONDVAR_DATA
  Condition Variables, 232
_EVENTSOURCE_DATA
  Event Flags, 245
_GUARDEDMEMORYPOOL_DATA
  Memory Pools, 402
_MAILBOX_DATA
  Mailboxes, 329
_MEMORYPOOL_DATA
  Memory Pools, 401
_MUTEX_DATA
  Mutexes, 217
PIPE_DATA
  Pipes, 346
_SEMAPHORE_DATA
  Counting Semaphores, 199
_THREADS_QUEUE_DATA
  Threads, 156
__CH_STRINGIFY
  Scheduler, 100
__ch_delegate_fn0
  Delegate Threads, 355
__ch_delegate_fn1
  Delegate Threads, 356
__ch_delegate_fn2
  Delegate Threads, 356
__ch_delegate_fn3
  Delegate Threads, 356
__ch_delegate_fn4
  Delegate Threads, 358
_core_init
  Core Memory Manager, 385
_dbg_check_disable
  Checks and Assertions, 293
_dbg_check_enable
  Checks and Assertions, 294
_dbg_check_enter_isr
  Checks and Assertions, 295
_dbg_check_lock
  Checks and Assertions, 297
_dbg_check_lock_from_isr
  Checks and Assertions, 298
_dbg_check_suspend
  Checks and Assertions, 293
_dbg_check_unlock
  Checks and Assertions, 295
_dbg_check_unlock_from_isr
  Checks and Assertions, 296
__factory_init
  Dynamic Objects Factory, 459
_heap_init
  Memory Heaps, 395
_idle_thread
  System Management, 67
_oslib_init
  Version Numbers and Identification, 314
_scheduler_init
  Scheduler, 103
_stats_ctxswc
  Statistics, 309
_stats_increase_irq
  Statistics, 308
_stats_init
  Statistics, 308
_stats_start_measure_crit_isr
  Statistics, 310
_stats_start_measure_crit_thd
  Statistics, 309
_stats_stop_measure_crit_isr
  Statistics, 310
_stats_stop_measure_crit_thd
  Statistics, 310
_thread_init
  Threads, 160
_thread_memfill
  Threads, 161
_tm_init
  Time Measurement, 194
_trace_halt
  Tracing, 302
_trace_init
  Tracing, 301
_trace_isr_enter
  Tracing, 302
_trace_isr_leave
  Tracing, 302
<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
<th>Page</th>
<th>Section</th>
</tr>
</thead>
<tbody>
<tr>
<td>_trace_switch</td>
<td>Tracing</td>
<td>301</td>
<td></td>
</tr>
<tr>
<td>_vt_init</td>
<td>Virtual Timers</td>
<td>135</td>
<td></td>
</tr>
<tr>
<td>align</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>memory_pool_t</td>
<td></td>
<td>558</td>
<td></td>
</tr>
<tr>
<td>ALL_EVENTS</td>
<td>Event Flags</td>
<td>245</td>
<td></td>
</tr>
<tr>
<td>arg</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>thread_descriptor_t</td>
<td></td>
<td>564</td>
<td></td>
</tr>
<tr>
<td>Base Kernel Services</td>
<td></td>
<td>54</td>
<td></td>
</tr>
<tr>
<td>basemem</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>memcore_t</td>
<td></td>
<td>554</td>
<td></td>
</tr>
<tr>
<td>best</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>time_measurement_t</td>
<td></td>
<td>566</td>
<td></td>
</tr>
<tr>
<td>Binary Semaphores</td>
<td></td>
<td>317</td>
<td></td>
</tr>
<tr>
<td>_BSEMAPHORE_DATA</td>
<td></td>
<td>317</td>
<td></td>
</tr>
<tr>
<td>binary_semaphore_t</td>
<td></td>
<td>318</td>
<td></td>
</tr>
<tr>
<td>BSEMAPHORE_DECL</td>
<td></td>
<td>318</td>
<td></td>
</tr>
<tr>
<td>chBSemGetStateI</td>
<td></td>
<td>326</td>
<td></td>
</tr>
<tr>
<td>chBSemObjectInit</td>
<td></td>
<td>318</td>
<td></td>
</tr>
<tr>
<td>chBSemReset</td>
<td></td>
<td>324</td>
<td></td>
</tr>
<tr>
<td>chBSemResetI</td>
<td></td>
<td>323</td>
<td></td>
</tr>
<tr>
<td>chBSemSignal</td>
<td></td>
<td>325</td>
<td></td>
</tr>
<tr>
<td>chBSemSignalI</td>
<td></td>
<td>325</td>
<td></td>
</tr>
<tr>
<td>chBSemWait</td>
<td></td>
<td>319</td>
<td></td>
</tr>
<tr>
<td>chBSemWaitS</td>
<td></td>
<td>320</td>
<td></td>
</tr>
<tr>
<td>chBSemWaitTimeout</td>
<td></td>
<td>322</td>
<td></td>
</tr>
<tr>
<td>chBSemWaitTimeoutS</td>
<td></td>
<td>321</td>
<td></td>
</tr>
<tr>
<td>binary_semaphore_t</td>
<td></td>
<td>318</td>
<td></td>
</tr>
<tr>
<td>BSEMAPHORE_DECL</td>
<td></td>
<td>318</td>
<td></td>
</tr>
<tr>
<td>buf_list</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ch_objects_factory</td>
<td></td>
<td>499</td>
<td></td>
</tr>
<tr>
<td>buffer</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ch_trace_buffer_t</td>
<td></td>
<td>529</td>
<td></td>
</tr>
<tr>
<td>mailbox_t</td>
<td></td>
<td>552</td>
<td></td>
</tr>
<tr>
<td>pipe_t</td>
<td></td>
<td>560</td>
<td></td>
</tr>
<tr>
<td>cache_sem</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ch_objects_cache</td>
<td></td>
<td>496</td>
<td></td>
</tr>
<tr>
<td>ch</td>
<td>Scheduler</td>
<td>114</td>
<td></td>
</tr>
<tr>
<td>ch.h</td>
<td></td>
<td>569</td>
<td></td>
</tr>
<tr>
<td>ch_binary_semaphore</td>
<td></td>
<td>477</td>
<td></td>
</tr>
<tr>
<td>CH_CFG_CONTEXT_SWITCH_HOOK</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Options</td>
<td></td>
<td>49</td>
<td></td>
</tr>
<tr>
<td>CH_CFG_FACTORY_GENERIC_BUFFERS</td>
<td>Dynamic Objects Factory, 456</td>
<td>Options</td>
<td>44</td>
</tr>
<tr>
<td>CH_CFG_FACTORY_MAILBOXES</td>
<td>Dynamic Objects Factory, 457</td>
<td>Options</td>
<td>45</td>
</tr>
<tr>
<td>CH_CFG_FACTORY_MAX_NAMES_LENGTH</td>
<td>Dynamic Objects Factory, 456</td>
<td>Options</td>
<td>44</td>
</tr>
<tr>
<td>Identifier</td>
<td>Location</td>
<td></td>
<td></td>
</tr>
<tr>
<td>---------------------------</td>
<td>---------------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>System Management</td>
<td>61</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ch_jobs_queue</td>
<td>489</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ch_list_init</td>
<td>491</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ch_list_notempty</td>
<td>83</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ch_list_pop</td>
<td>85</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ch_list_push</td>
<td>84</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ch_list</td>
<td>83</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ch_memcore</td>
<td>392</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ch_mutex</td>
<td>492</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ch_objects_cache</td>
<td>494</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ch_objects_factory</td>
<td>497</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ch_object</td>
<td>500</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ch_oc_hash_next</td>
<td>502</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ch_oc_hash_prev</td>
<td>503</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ch_oc_iru_next</td>
<td>504</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ch_oc_iru_prev</td>
<td>504</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ch_oc_object</td>
<td>505</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ch_pqueue_init</td>
<td>88</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ch_pqueue_insert_ahead</td>
<td>90</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ch_pqueue_insert_behind</td>
<td>89</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ch_pqueue</td>
<td>88</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ch_pqueue</td>
<td>89</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ch_pqueue_insert</td>
<td>88</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ch_pqueue_insert</td>
<td>89</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ch_pqueue</td>
<td>89</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ch_pqueue</td>
<td>88</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ch_pqueue</td>
<td>89</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ch_pqueue</td>
<td>89</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

ChibiOS/RT
ch_pqueue_remove_highest
  Lists and Queues, 89
ch_priority_queue, 508
  next, 508
  prev, 508
ch_priority_queue_t
  Lists and Queues, 83
ch_queue, 509
  next, 509
  prev, 510
CH_QUEUE_DECL
  Lists and Queues, 82
ch_queue_dequeue
  Lists and Queues, 88
ch_queue_fifo_remove
  Lists and Queues, 87
ch_queue_init
  Lists and Queues, 85
ch_queue_insert
  Lists and Queues, 86
ch_queue_isempty
  Lists and Queues, 86
ch_queue_lifo_remove
  Lists and Queues, 87
ch_queue_notempty
  Lists and Queues, 86
ch_queue_t
  Lists and Queues, 83
ch_registered_static_object, 510
  element, 511
  objp, 511
ch_sch_prio_insert
  Scheduler, 102
ch_semaphore, 511
  cnt, 512
  queue, 512
CH_STATE_CURRENT
  Scheduler, 96
CH_STATE_FINAL
  Scheduler, 98
CH_STATE_NAMES
  Scheduler, 99
CH_STATEQUEUED
  Scheduler, 97
CH_STATE_READY
  Scheduler, 96
CH_STATE_SLEEPING
  Scheduler, 97
CH_STATE_SNDMSG
  Scheduler, 98
CH_STATE_SNDMSGQ
  Scheduler, 98
CH_STATE_SUSPENDED
  Scheduler, 96
CH_STATE_WTANDEVT
  Scheduler, 98
CH_STATE_WTCASE
  Scheduler, 97
CH_STATE_WTCOND
  Scheduler, 97
CH_STATE_WTEXIT
  Scheduler, 97
CH_STATE_WTMSG
  Scheduler, 98
CH_STATE_WTMTX
  Scheduler, 98
CH_STATE_WTOREVT
  Scheduler, 98
CH_STATE_WTSEMS
  Scheduler, 97
CH_STATE_WTSTART
  Scheduler, 96
ch_system, 513
  dbg, 514
  kernel_stats, 515
  mainthread, 514
  rlist, 514
  tm, 514
  vlist, 514
ch_system_debug, 515
  isr_cnt, 517
  lock_cnt, 517
  panic_msg, 517
  trace_buffer, 517
ch_system_t
  Scheduler, 102
ch_thread, 518
  ctx, 520
  epending, 525
  ewmask, 524
  exitcode, 522
  flags, 521
  list, 520
  mpool, 525
  msgqueue, 525
  mtxlist, 525
  name, 521
  newer, 520
  older, 521
  pqueue, 520
  queue, 520
  rdymsg, 522
  realprio, 525
  refs, 522
  sentmsg, 523
  state, 521
  stats, 525
  ticks, 522
  time, 522
  u, 524
  wabase, 521
  waiting, 524
  wtmtxp, 524
  wtobjp, 523
  wtsemp, 523
  wttrp, 523
ch_threads_queue, 526
  queue, 526
ch_trace_buffer_t, 527
  buffer, 529
  ptr, 529
  size, 529
  suspended, 529
ch_trace_event_t, 530
  halt, 533
  isr, 533
  name, 532
  ntp, 532
  reason, 533
  rtstamp, 532
  state, 531
  sw, 532
  type, 531
  up1, 533
  up2, 533
  user, 533
  wtobjp, 532
CH_VERSION
  Release Information, 20
CH_VERSION_MONTH
  Release Information, 21
CH_VERSION_NICKNAME
  Release Information, 21
CH_VERSION_PATCH
  Release Information, 21
CH_VERSION_STABLE
  Release Information, 20
CH_VERSION_YEAR
  Release Information, 21
ch_virtual_timer, 534
  dist, 535
  func, 535
  par, 535
ch_virtual_timers_list, 535
  dist, 536
  lasttime, 536
  systime, 536
chalign.h, 570
chbsem.h, 571
chBSemGetStatelt
  Binary Semaphores, 326
chBSemObjectName
  Binary Semaphores, 318
chBSemReset
  Binary Semaphores, 324
chBSemResetIt
  Binary Semaphores, 323
chBSemSignal
  Binary Semaphores, 325
chBSemSignalIt
  Binary Semaphores, 325
chBSemWait
  Binary Semaphores, 319
chBSemWaitIt
  Binary Semaphores, 320
chBSemWaitTimeout
  Binary Semaphores, 322
chBSemWaitTimeoutIt
  Binary Semaphores, 321
chCacheGetObject
  Objects Caches, 448
chCacheObjectName
  Objects Caches, 447
chCacheReadObject
  Objects Caches, 449
chCacheReleaseObject
  Objects Caches, 451
chCacheReleaseObjectName
  Objects Caches, 449
chCacheWriteObject
  Objects Caches, 450
chchecks.h, 572
chcond.c, 572
chcond.h, 573
chCondBroadcast
  Condition Variables, 235
chCondBroadcastIt
  Condition Variables, 236
chCondObjectName
  Condition Variables, 232
chCondSignal
  Condition Variables, 233
chCondSignalIt
  Condition Variables, 234
chCondWait
  Condition Variables, 237
chCondWaitIt
  Condition Variables, 238
chCondWaitTimeout
  Condition Variables, 239
chCondWaitTimeoutIt
  Condition Variables, 240
chconf.h, 574
chCoreAlloc
  Core Memory Manager, 391
chCoreAllocAligned
  Core Memory Manager, 390
chCoreAllocAlignedIt
  Core Memory Manager, 389
chCoreAllocAlignedWithOffset
  Core Memory Manager, 384
chCoreAllocAlignedWithOffsetIt
  Core Memory Manager, 383
chCoreAllocFromBase
  Core Memory Manager, 387
chCoreAllocFromBaseIt
  Core Memory Manager, 385
chCoreAllocFromTop
  Core Memory Manager, 388
chCoreAllocFromTopIt
  Core Memory Manager, 386
chCoreAllocI
  Core Memory Manager, 391
INDEX

chCoreGetStatusX
Core Memory Manager, 389
chCustomer.h, 577
chDbgAssert
Checks and Assertions, 292
chDbgCheck
Checks and Assertions, 291
chDbgCheckClassI
Checks and Assertions, 297
chDbgCheckClassS
Checks and Assertions, 298
chDbgResumeTrace
Tracing, 306
chDbgResumeTraceI
Tracing, 306
chDbgSuspendTrace
Tracing, 305
chDbgSuspendTraceI
Tracing, 304
chDbgWriteTrace
Tracing, 303
chDbgWriteTraceI
Tracing, 303
chdebug.c, 578
chdebug.h, 578
chdebug_t, 537
identifier, 538
off_ctx, 539
off_flags, 540
off_name, 540
off_newer, 540
off_older, 540
off_preempt, 541
off_prio, 539
off_refs, 541
off_state, 540
off_stklimit, 540
off_time, 541
ptrsize, 539
size, 538
threadsize, 539
timesize, 539
version, 539
zero, 538
chDelegateCallDirect0
Delegate Threads, 360
chDelegateCallDirect1
Delegate Threads, 361
chDelegateCallDirect2
Delegate Threads, 362
chDelegateCallDirect3
Delegate Threads, 363
chDelegateCallDirect4
Delegate Threads, 364
chDelegateCallVeneer
Delegate Threads, 358
chDelegateDispatch
Delegate Threads, 359
chDelegateDispatchTimeout
Delegate Threads, 359
chdelegates.c, 579
chdelegates.h, 580
chdynamic.c, 581
chdynamic.h, 581
Checks, 52
Checks and Assertions, 290
_dbg_check_disable, 293
_dbg_check_enable, 294
_dbg_check_enter_isr, 296
_dbg_check_leave_isr, 297
_dbg_check_lock, 294
_dbg_check_lock_from_isr, 295
_dbg_check_suspend, 293
_dbg_check_unlock, 295
_dbg_check_unlock_from_isr, 296
CH_DBG_STACK_FILL_VALUE, 291
chDbgAssert, 292
chDbgCheck, 291
chDbgCheckClassI, 297
chDbgCheckClassS, 298
chevents.c, 582
chevents.h, 583
chEvtAddEvents
Event Flags, 249
chEvtAddEventsI
Event Flags, 266
chEvtBroadcast
Event Flags, 264
chEvtBroadcastFlags
Event Flags, 254
chEvtBroadcastFlagsI
Event Flags, 250
chEvtBroadcastI
Event Flags, 265
chEvtDispatch
Event Flags, 255
chEvtGetAndClearEvents
Event Flags, 249
chEvtGetAndClearEventsI
Event Flags, 248
chEvtGetAndClearFlags
Event Flags, 251
chEvtGetAndClearFlagsI
Event Flags, 255
chEvtGetEventsX
Event Flags, 266
chEvtIsListeningI
Event Flags, 264
chEvtObjectInit
Event Flags, 262
chEvtRegister
Event Flags, 262
chEvtRegisterMask
Event Flags, 263
chEvtRegisterMaskWithFlags
Event Flags, 246
INDEX

chGuardedPoolAllocTimeout
Memory Pools, 412
chGuardedPoolAllocTimeoutS
Memory Pools, 411
chGuardedPoolFree
Memory Pools, 413
chGuardedPoolFreeI
Memory Pools, 419
chGuardedPoolFreeS
Memory Pools, 420
chGuardedPoolGetCounterI
Memory Pools, 417
chGuardedPoolLoadArray
Memory Pools, 410
chGuardedPoolObjectInit
Memory Pools, 417
chGuardedPoolObjectInitAligned
Memory Pools, 409
chHeapAlloc
Memory Heaps, 397
chHeapAllocAligned
Memory Heaps, 396
chHeapFree
Memory Heaps, 396
chHeapGetSize
Memory Heaps, 398
chHeapObjectInit
Memory Heaps, 395
chHeapStatus
Memory Heaps, 397
chJobDispatch
Jobs Queues, 378
chJobDispatchTimeout
Jobs Queues, 379
chJobGet
Jobs Queues, 368
chJobGetI
Jobs Queues, 369
chJobGetTimeout
Jobs Queues, 371
chJobGetTimeoutS
Jobs Queues, 370
chJobObjectInit
Jobs Queues, 367
chJobPost
Jobs Queues, 374
chJobPostAhead
Jobs Queues, 377
chJobPostAheadI
Jobs Queues, 375
chJobPostAheadS
Jobs Queues, 376
chJobPostI
Jobs Queues, 372
chJobPostS
Jobs Queues, 373
chjobs.h, 589
chlib.h, 590
chlicense.h, 591
chlists.h, 592
chMBFetchI
Mailboxes, 341
chMBFetchTimeout
Mailboxes, 339
chMBFetchTimeoutS
Mailboxes, 339
chMBGetFreeCountI
Mailboxes, 340
chMBGetSizel
Mailboxes, 342
chMBGetUsedCountI
Mailboxes, 343
chMBOBJECTCountInit
Mailboxes, 330
chmboxes.c, 594
chmboxes.h, 594
chMBpeekI
Mailboxes, 344
chMBPostAheadI
Mailboxes, 338
chMBPostAheadTimeout
Mailboxes, 336
chMBPostAheadTimeoutS
Mailboxes, 337
chMBPostI
Mailboxes, 335
chMBPostTimeout
Mailboxes, 333
chMBPostTimeoutS
Mailboxes, 334
chMBReset
Mailboxes, 331
chMBResetI
Mailboxes, 332
chMBResumeX
Mailboxes, 345
chmemcore.c, 596
chmemcore.h, 596
chmemheaps.c, 597
chmemheaps.h, 598
chmempools.c, 599
chmempools.h, 600
chmsg.c, 601
chmsg.h, 602
chMsgGet
Synchronous Messages, 276
chMsgGetPendingI
Synchronous Messages, 275
chMsgPoll
Synchronous Messages, 274
chMsgPollI
Synchronous Messages, 270
chMsgRelease
Synchronous Messages, 271
chMsgReleaseS
Synchronous Messages, 277
chMsgSend
  Synchronous Messages, 268
chMsgWait
  Synchronous Messages, 272
chMsgWaitS
  Synchronous Messages, 268
chMsgWaitTimeout
  Synchronous Messages, 273
chMsgWaitTimeoutS
  Synchronous Messages, 269
chmtx.c, 602
chmtx.h, 603
chMtxGetNextMutexX
  Mutexes, 230
chMtxGetOwnerI
  Mutexes, 229
chMtxLock
  Mutexes, 220
chMtxTryLock
  Mutexes, 222
chMtxTryLockS
  Mutexes, 223
chMtxUnlock
  Mutexes, 224
chMtxUnlockAll
  Mutexes, 227
chMtxUnlockAllS
  Mutexes, 226
chMtxUnlockS
  Mutexes, 225
chobjcaches.c, 604
chobjcaches.h, 605
chobjfifos.h, 607
chPipeGetFreeCount
  Pipes, 352
chPipeGetSize
  Pipes, 351
chPipeGetUsedCount
  Pipes, 351
chPipeObjectInit
  Pipes, 349
chPipeReadTimeout
  Pipes, 350
chPipeReset
  Pipes, 349
chPipeResume
  Pipes, 352
chipipes.c, 608
chipipes.h, 609
chPipeWriteTimeout
  Pipes, 349
chPoolAdd
  Memory Pools, 415
chPoolAddI
  Memory Pools, 416
chPoolAlloc
  Memory Pools, 406
chPoolAllocI
  Memory Pools, 405
chPoolFree
  Memory Pools, 408
chPoolFreeI
  Memory Pools, 407
chPoolLoadArray
  Memory Pools, 404
chPoolObjectInit
  Memory Pools, 414
chPoolObjectInitAligned
  Memory Pools, 403
chRegFindThreadByName
  Registry, 284
chRegFindThreadByPointer
  Registry, 285
chRegFindThreadByWorkingArea
  Registry, 286
chRegFirstThread
  Registry, 283
chRegGetThreadNameX
  Registry, 287
chregistry.c, 610
chregistry.h, 611
chRegNextThread
  Registry, 283
chRegSetThreadName
  Registry, 287
chRegSetThreadNameX
  Registry, 288
chrestrictions.h, 611
chSchCanYieldS
  Scheduler, 112
chSchDoReschedule
  Scheduler, 110
chSchDoRescheduleAhead
  Scheduler, 109
chSchDoRescheduleBehind
  Scheduler, 109
chSchDoYieldS
  Scheduler, 112
chSchGoSleepS
  Scheduler, 105
chSchGoSleepTimeoutS
  Scheduler, 106
chSchlsPreemptionRequired
  Scheduler, 108
chSchlsRescRequiredI
  Scheduler, 111
chSchPreemption
  Scheduler, 113
<table>
<thead>
<tr>
<th>Function</th>
<th>Pages</th>
<th>Section</th>
<th>Pages</th>
<th>Section</th>
</tr>
</thead>
<tbody>
<tr>
<td>chThdGetPriorityX</td>
<td>186</td>
<td>Threads</td>
<td>186</td>
<td>Threads</td>
</tr>
<tr>
<td>chThdGetSelfX</td>
<td>186</td>
<td>Threads</td>
<td>186</td>
<td>Threads</td>
</tr>
<tr>
<td>chThdGetTicksX</td>
<td>186</td>
<td>Threads</td>
<td>186</td>
<td>Threads</td>
</tr>
<tr>
<td>chThdGetWorkingAreaX</td>
<td>187</td>
<td>Threads</td>
<td>187</td>
<td>Threads</td>
</tr>
<tr>
<td>chThdQueueIsEmptyI</td>
<td>190</td>
<td>Threads</td>
<td>190</td>
<td>Threads</td>
</tr>
<tr>
<td>chThdQueueObjectInit</td>
<td>190</td>
<td>Threads</td>
<td>190</td>
<td>Threads</td>
</tr>
<tr>
<td>chThdRelease</td>
<td>169</td>
<td>Threads</td>
<td>169</td>
<td>Threads</td>
</tr>
<tr>
<td>chThdResume</td>
<td>182</td>
<td>Threads</td>
<td>182</td>
<td>Threads</td>
</tr>
<tr>
<td>chThdResumeI</td>
<td>181</td>
<td>Threads</td>
<td>181</td>
<td>Threads</td>
</tr>
<tr>
<td>chThdResumeS</td>
<td>181</td>
<td>Threads</td>
<td>181</td>
<td>Threads</td>
</tr>
<tr>
<td>chThdSetPriority</td>
<td>173</td>
<td>Threads</td>
<td>173</td>
<td>Threads</td>
</tr>
<tr>
<td>chThdShouldTerminateX</td>
<td>188</td>
<td>Threads</td>
<td>188</td>
<td>Threads</td>
</tr>
<tr>
<td>chThdSleep</td>
<td>175</td>
<td>Threads</td>
<td>175</td>
<td>Threads</td>
</tr>
<tr>
<td>chThdSleepMicroseconds</td>
<td>159</td>
<td>Threads</td>
<td>159</td>
<td>Threads</td>
</tr>
<tr>
<td>chThdSleepMilliseconds</td>
<td>158</td>
<td>Threads</td>
<td>158</td>
<td>Threads</td>
</tr>
<tr>
<td>chThdSleepS</td>
<td>189</td>
<td>Threads</td>
<td>189</td>
<td>Threads</td>
</tr>
<tr>
<td>chThdSleepSeconds</td>
<td>158</td>
<td>Threads</td>
<td>158</td>
<td>Threads</td>
</tr>
<tr>
<td>chThdSleepUntil</td>
<td>176</td>
<td>Threads</td>
<td>176</td>
<td>Threads</td>
</tr>
<tr>
<td>chThdSleepUntilWindowed</td>
<td>177</td>
<td>Threads</td>
<td>177</td>
<td>Threads</td>
</tr>
<tr>
<td>chThdStart</td>
<td>167</td>
<td>Threads</td>
<td>167</td>
<td>Threads</td>
</tr>
<tr>
<td>chThdStartI</td>
<td>188</td>
<td>Threads</td>
<td>188</td>
<td>Threads</td>
</tr>
<tr>
<td>chThdSuspend</td>
<td>179</td>
<td>Threads</td>
<td>179</td>
<td>Threads</td>
</tr>
<tr>
<td>chThdSuspendTimeoutS</td>
<td>180</td>
<td>Threads</td>
<td>180</td>
<td>Threads</td>
</tr>
<tr>
<td>chThdTerminate</td>
<td>174</td>
<td>Threads</td>
<td>174</td>
<td>Threads</td>
</tr>
<tr>
<td>chThdTerminatedX</td>
<td>187</td>
<td>Threads</td>
<td>187</td>
<td>Threads</td>
</tr>
<tr>
<td>chThdWait</td>
<td>172</td>
<td>Threads</td>
<td>172</td>
<td>Threads</td>
</tr>
<tr>
<td>chThdYield</td>
<td>178</td>
<td>Threads</td>
<td>178</td>
<td>Threads</td>
</tr>
<tr>
<td>chthreads.c</td>
<td>622</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>chthreads.h</td>
<td>624</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ctime.h</td>
<td>626</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>chTimeAddX</td>
<td></td>
<td>Time and Intervals</td>
<td>125</td>
<td></td>
</tr>
<tr>
<td>chTimeDiffX</td>
<td></td>
<td>Time and Intervals</td>
<td>126</td>
<td></td>
</tr>
<tr>
<td>chTimeI2MS</td>
<td></td>
<td>Time and Intervals</td>
<td>124</td>
<td></td>
</tr>
<tr>
<td>chTimeI2S</td>
<td></td>
<td>Time and Intervals</td>
<td>124</td>
<td></td>
</tr>
<tr>
<td>chTimeI2US</td>
<td></td>
<td>Time and Intervals</td>
<td>125</td>
<td></td>
</tr>
<tr>
<td>chTimeInRangeX</td>
<td></td>
<td>Time and Intervals</td>
<td>126</td>
<td></td>
</tr>
<tr>
<td>chTimeMS2I</td>
<td></td>
<td>Time and Intervals</td>
<td>123</td>
<td></td>
</tr>
<tr>
<td>chTimeS2I</td>
<td></td>
<td>Time and Intervals</td>
<td>122</td>
<td></td>
</tr>
<tr>
<td>chTimeUS2I</td>
<td></td>
<td>Time and Intervals</td>
<td>123</td>
<td></td>
</tr>
<tr>
<td>chtm.c</td>
<td>628</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>chtm.h</td>
<td>628</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>chTMChainMeasurementToX</td>
<td></td>
<td>Time Measurement</td>
<td>195</td>
<td></td>
</tr>
<tr>
<td>chTMObjectInit</td>
<td></td>
<td>Time Measurement</td>
<td>194</td>
<td></td>
</tr>
<tr>
<td>chTMStartMeasurementX</td>
<td></td>
<td>Time Measurement</td>
<td>194</td>
<td></td>
</tr>
<tr>
<td>chTMStopMeasurementX</td>
<td></td>
<td>Time Measurement</td>
<td>195</td>
<td></td>
</tr>
<tr>
<td>chtrace.c</td>
<td>629</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>chtrace.h</td>
<td>630</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>chversion.h</td>
<td>631</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>chvt.c</td>
<td>632</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>chvt.h</td>
<td>633</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>chVTDoResetI</td>
<td></td>
<td>Virtual Timers</td>
<td>137</td>
<td></td>
</tr>
<tr>
<td>chVTDoSetI</td>
<td></td>
<td>Virtual Timers</td>
<td>136</td>
<td></td>
</tr>
<tr>
<td>chVTDoTickI</td>
<td></td>
<td>Virtual Timers</td>
<td>139</td>
<td></td>
</tr>
<tr>
<td>chVTGetRemainingIntervalI</td>
<td></td>
<td>Virtual Timers</td>
<td>138</td>
<td></td>
</tr>
<tr>
<td>chVTGetSystemTime</td>
<td></td>
<td>Virtual Timers</td>
<td>141</td>
<td></td>
</tr>
<tr>
<td>chVTGetSystemTimeX</td>
<td></td>
<td>Virtual Timers</td>
<td>141</td>
<td></td>
</tr>
<tr>
<td>chVTGetTimersStateI</td>
<td></td>
<td>Virtual Timers</td>
<td>145</td>
<td></td>
</tr>
<tr>
<td>chVTIsArmed</td>
<td></td>
<td>Virtual Timers</td>
<td>147</td>
<td></td>
</tr>
<tr>
<td>chVTIsArmedI</td>
<td></td>
<td>Virtual Timers</td>
<td>146</td>
<td></td>
</tr>
<tr>
<td>chVTIsSystemTimeWithin</td>
<td></td>
<td>Virtual Timers</td>
<td>144</td>
<td></td>
</tr>
<tr>
<td>chVTIsSystemTimeXWithin</td>
<td></td>
<td>Virtual Timers</td>
<td>143</td>
<td></td>
</tr>
<tr>
<td>chVTOBJECTINIT</td>
<td></td>
<td>Virtual Timers</td>
<td>140</td>
<td></td>
</tr>
<tr>
<td>chVTReset</td>
<td></td>
<td>Virtual Timers</td>
<td>149</td>
<td></td>
</tr>
</tbody>
</table>
chVTReset
   Virtual Timers, 148
chVTSet
   Virtual Timers, 151
chVTSetI
   Virtual Timers, 150
chVTTimeElapsedSinceX
   Virtual Timers, 142
cmtx
   pipe_t, 561
cnt
   ch_mutex, 493
   ch_semaphore, 512
   mailbox_t, 553
   pipe_t, 561
Complex Services, 425
Condition Variables, 231
   _CONDVAR_DATA, 232
   chCondBroadcast, 235
   chCondBroadcastI, 236
   chCondObjectInit, 232
   chCondSignal, 233
   chCondSignalI, 234
   chCondWait, 237
   chCondWaitS, 238
   chCondWaitTimeout, 239
   chCondWaitTimeoutS, 240
   condition_variable_t, 232
   CONDVAR_DECL, 232
condition_variable, 541
   queue, 542
condition_variable_t
   Condition Variables, 232
   CONDVAR_DECL
   Condition Variables, 232
Configuration, 31
Core Memory Manager, 382
   _core_init, 385
   CH_CFG_MEMCORE_SIZE, 383
   ch_memcore, 392
   chCoreAlloc, 391
   chCoreAllocAligned, 390
   chCoreAllocAlignedI, 389
   chCoreAllocAlignedWithOffset, 384
   chCoreAllocAlignedWithOffsetI, 383
   chCoreAllocFromBase, 387
   chCoreAllocFromBaseI, 385
   chCoreAllocFromTop, 388
   chCoreAllocFromTopI, 386
   chCoreAllocI, 391
   chCoreGetStatusX, 389
   memgetfunc2_t, 384
   memgetfunc3_t, 384
Counting Semaphores, 198
   _SEMAPHORE_DATA, 199
   chSemAddCounterI, 209
   chSemFastSignal, 213
   chSemFastWaitI, 213
chSemGetCounterI, 214
chSemObjectInit, 200
chSemReset, 211
chSemResetI, 212
chSemResetWithMessage, 201
chSemResetWithMessageI, 202
chSemSignal, 207
chSemSignalI, 208
chSemSignalWait, 210
chSemWait, 203
chSemWaitS, 204
chSemWaitTimeout, 205
chSemWaitTimeoutS, 206
SEMAPHORE_DECL, 200
semaphore_t, 200
ctx
   ch_thread, 520
   cumulative
   time_measurement_t, 566
curpp
   Scheduler, 100
Customer Information, 22
   CH_CUSTOMER_ID_CODE, 22
   CH_CUSTOMER_ID_STRING, 22
   CH_LICENSE, 23
dbg
   ch_system, 514
Debug, 289
default_heap
   Memory Heaps, 399
Delegate Threads, 354
   __ch_delegate_fn0, 355
   __ch_delegate_fn1, 356
   __ch_delegate_fn2, 356
   __ch_delegate_fn3, 356
   __ch_delegate_fn4, 358
   chDelegateCallDirect0, 360
   chDelegateCallDirect1, 361
   chDelegateCallDirect2, 362
   chDelegateCallDirect3, 363
   chDelegateCallDirect4, 364
   chDelegateCallVeneer, 358
   chDelegateDispatch, 359
   chDelegateDispatchTimeout, 359
   delegate_fn0, 355
   delegate_fn1, 355
   delegate_fn2, 355
   delegate_fn3, 355
   delegate_fn4, 355
   delegate_veneer, 355
   delegate_fn0, 355
   Delegate Threads, 355
delegate_fn1, 355
   Delegate Threads, 355
delegate_fn2, 355
   Delegate Threads, 355
delegate_fn3, 355
   Delegate Threads, 355
delegate_fn4, 355
   Delegate Threads, 355
delegate_fn4_t
  Delegate Threads, 355
delagate_veneer_t
  Delegate Threads, 355
delta
  ch_delta_list, 479
delta_list_t
  Scheduler, 101
dlist
  ch_virtual_timer, 535
  ch_virtual_timers_list, 536
dptr
  ch_oc_object, 507
dyn_element_t
  Dynamic Objects Factory, 458
dyn_list_t
  Dynamic Objects Factory, 458
dyn_mailbox_t
  Dynamic Objects Factory, 458
dyn_objects_fifo_t
  Dynamic Objects Factory, 459
dyn_pipe_t
  Dynamic Objects Factory, 459
dyn_semaphore_t
  Dynamic Objects Factory, 458
Dynamic Objects Factory, 453
  _factory_init, 459
  CH_CFG_FACTORY_GENERIC_BUFFERS, 456
  CH_CFG_FACTORY_MAILBOXES, 457
  CH_CFG_FACTORY_MAX_NAMES_LENGTH, 456
  CH_CFG_FACTORY_OBJ_FIFOS, 457
  CH_CFG_FACTORY_OBJECTS_REGISTRY, 456
  CH_CFG_FACTORYPIPES, 457, 458
  CH_CFG_FACTORY_SEMAPHORES, 456
  ch_factory, 476
  chFactoryCreateBuffer, 462
  chFactoryCreateMailbox, 466
  chFactoryCreateObjectsFIFO, 467
  chFactoryCreatePipe, 469
  chFactoryCreateSemaphore, 464
  chFactoryDuplicateReference, 472
  chFactoryFindBuffer, 463
  chFactoryFindMailbox, 466
  chFactoryFindObject, 461
  chFactoryFindObjectByPointer, 461
  chFactoryFindObjectBySize, 468
  chFactoryFindPipe, 470
  chFactoryFindSemaphore, 465
  chFactoryGetBuffer, 473
  chFactoryGetBufferSize, 473
  chFactoryGetMailbox, 474
  chFactoryGetObject, 472
  chFactoryGetObjectsFIFO, 475
  chFactoryGetPipe, 475
  chFactoryGetSemaphore, 474
  chFactoryRegisterObject, 460
  chFactoryReleaseBuffer, 464
  chFactoryReleaseMailbox, 467
  chFactoryReleaseObject, 462
  chFactoryReleaseObjectsFIFO, 469
  chFactoryReleasePipe, 470
  chFactoryReleaseSemaphore, 465
dyn_buffer_t
  Dynamic Objects Factory, 458
dyn_element_t
  Dynamic Objects Factory, 458
dyn_list_t
  Dynamic Objects Factory, 458
dyn_mailbox_t
  Dynamic Objects Factory, 459
dyn_objects_fifo_t
  Dynamic Objects Factory, 459
  ch_dyn_mailbox, 482
  ch_dyn_object, 484
  ch_dyn_objects_fifo, 485
  ch_dyn_pipe, 487
  ch_dyn_semaphore, 488
  ch_registered_stack_object, 511
  ch_thread, 525
Event Flags, 243
  _EVENTSOURCE_DATA, 245
  ALL_EVENTS, 245
  chEvtAddEvents, 249
  chEvtAddEventsI, 266
  chEvtBroadcast, 264
  chEvtBroadcastFlags, 254
  chEvtBroadcastFlagsI, 250
  chEvtBroadcastsl, 265
  chEvtDispatch, 255
  chEvtGetAndClearEvents, 249
  chEvtGetAndClearEventsI, 248
  chEvtGetAndClearFlags, 251
  chEvtGetAndClearFlagsI, 255
  chEvtGetEventsX, 266
  chEvtIsListeningI, 264
  chEvtObjectInit, 262
  chEvtRegister, 263
  chEvtRegisterMask, 263
  chEvtRegisterMaskWithFlags, 246
  chEvtSignal, 252
  chEvtSignalAll, 253
  chEvtUnregister, 247
  chEvtWaitAll, 258
  chEvtWaitAllTimeout, 261
  chEvtWaitAny, 257
  chEvtWaitAnyTimeout, 260
  chEvtWaitOne, 256
  chEvtWaitOneTimeout, 259
  EVENT_MASK, 245
event_source_t, 246
EVENTSOURCE_DECL, 246
evhandler_t, 246
event_listener, 542
events, 543
flags, 543
listener, 543
next, 543
wflags, 544
EVENT_MASK
Event Flags, 245
event_source, 544
next, 545
event_source_t
Event Flags, 246
events
event_listener, 543
EVENTSOURCE_DECL
Event Flags, 246
evhandler_t
Event Flags, 246
ewmask
ch_thread, 524
exitcode
ch_thread, 522
FALSE
Version Numbers and Identification, 28
fifo
ch_dyn_objects_fifo, 485
fifo_list
ch_objects_factory, 499
firstprio
Scheduler, 100
flags
ch_thread, 521
event_listener, 543
free
ch_jobs_queue, 490
ch_objects_fifo, 501
func
ch_virtual_timer, 535
funcp
thread_descriptor_t, 564
guarded_memory_pool_t, 546
pool, 547
sem, 547
GUARDEDMEMORYPOOL_DECL
Memory Pools, 403
halt
ch_trace_event_t, 533
hash_get_s
Objects Caches, 445
hash_next
ch_oc_hash_header, 502
ch_oc_lru_header, 504
ch_oc_object, 506
hashn
ch_objects_cache, 495
hashp
ch_objects_cache, 495
header
memory_heap, 556
heap
heap_header, 548
heap_header, 548
next, 548
pages, 548
size, 548
heap_header_t
Memory Heaps, 395
HIGHPRIO
Scheduler, 96
identifier
chdebug_t, 538
IDLEPRIO
Scheduler, 95
isr
ch_trace_event_t, 533
isr_cnt
ch_system_debug, 517
job_descriptor_t
Jobs Queues, 367
job_function_t
Jobs Queues, 367
jobarg
ch_job_descriptor, 489
jobfunc
ch_job_descriptor, 489
Jobs Queues, 366
chJobDispatch, 378
chJobDispatchTimeout, 379
chJobGet, 368
chJobGetI, 369
chJobGetTimeout, 371
chJobGetTimeoutS, 370
chJobObjectInit, 367
chJobPost, 374
chJobPostAhead, 377
chJobPostAheadI, 375
chJobPostAheadS, 376
chJobPostI, 372
chJobPostS, 373
job_descriptor_t, 367
job_function_t, 367
jobs_queue_t
MSG_JOB_NULL, 367
Jobs Queues, 367
kernel_stats
c
kernel_stats_t, 549
m_crit_isr, 550
m_crit_thd, 550
n_ctxswc, 550
n_irq, 550

time_measurement_t, 566

c
License Settings, 24
CH_LICENSE_FEATURES, 25
CH_LICENSE_ID_CODE, 25
CH_LICENSE_ID_STRING, 25
CH_LICENSE_MAX_DEPLOY, 25
CH_LICENSE_MODIFIABLE_CODE, 25
CH_LICENSE_TYPE_STRING, 24

list
c

listener
event_listener, 543

Lists and Queues, 81
_CH_QUEUE_DATA, 82
ch_list_init, 83
ch_list_isempty, 83
ch_list_notempty, 84
ch_list_pop, 85
ch_list_push, 84
ch_list_t, 83
ch_pqueue_init, 88
ch_pqueue_insert_ahead, 90
ch_pqueue_insert_behind, 89
ch_pqueue_remove_highest, 89
ch_priority_queue_t, 83
CH_QUEUE_DECL, 82
ch_queue_dequeue, 88
ch_queue_fifo_remove, 87
ch_queue_init, 85
ch_queue_insert, 86
ch_queue_isempty, 86
ch_queue_isnotempty, 86
ch_queue_remove, 87
ch_queue_notempty, 86
ch_queue_t, 83

lock_cnt
c

LOWPRIO
Scheduler, 95

lru
c

lru_get_last_s
Objects Caches, 446

lru_next
c

lru_prev
c

mailbox

mailbox_t, 551
buffer, 552
cnt, 553
qr, 553
qw, 553
rdptr, 552
reset, 553
top, 552
wrptr, 552

Mailboxes, 328
_MAILBOX_DATA, 329
chMBFetchI, 341
chMBFetchTimeout, 339
chMBFetchTimeoutS, 340
chMBGetFreeCountI, 343
chMBGetUsedCountI, 343
chMOBJECTINIT, 330
chMBPeekI, 344
chMBPostAheadI, 338
chMBPostAheadTimeout, 336
chMBPostAheadTimeoutS, 337
chMBPostI, 335
chMBPostTimeout, 333
chMBPostTimeoutS, 334
chMBReset, 331
chMBResetI, 332
chMBResumeX, 345
MAILBOX_DECL, 330

mainthread
c

mbx
c

mbx_list
c

memcore
basemem, 554
topmem, 554
ch_delta_list, 479
ch_dyn_element, 480
ch_list, 491
ch_mutex, 493
ch_priority_queue, 508
ch_queue, 509
event_listener, 543
event_source, 545
heap_header, 548
memory_pool_t, 557
pool_header, 563

NOPRIOR
Scheduler, 95
NORMALPRIOR
Scheduler, 96

ntp
ch_trace_event_t, 532

obj_flags
ch_oc_object, 507

obj_group
ch_oc_object, 506

obj_key
ch_oc_object, 507

obj_list
ch_objects_factory, 498

obj_pool
ch_objects_factory, 498

obj_sem
ch_oc_object, 507

object_size
memory_pool_t, 558

Objects Caches, 443
chCacheGetObject, 448
chCacheObjectInit, 447
chCacheReadObject, 449
chCacheReleaseObject, 451
chCacheReleaseObjectI, 449
chCacheWriteObject, 450
hash_get_s, 445
lru_get_last_s, 446
objects_cache_t, 445
oc_flags_t, 444
oc_hash_header_t, 444
oc_lru_header_t, 444
oc_object_t, 444
oc_readf_t, 445
oc_writef_t, 445

Objects FIFOs, 426
chFifoObjectInit, 428
chFifoObjectInitAligned, 427
chFifoReceiveObject, 439
chFifoReceiveObjectTimeout, 441
chFifoReceiveObjectTimeoutS, 440
chFifoReturnObject, 433
chFifoReturnObjectI, 432
chFifoReturnObjectS, 432
chFifoSendObject, 436
chFifoSendObjectAhead, 439

objects_cache_t
Objects Caches, 445

objects_factory_t
Dynamic Objects Factory, 459

objects_fifo_t
Objects FIFOs, 427

objn
ch_objects_cache, 495

objp
ch_registered_static_object, 511

objsz
ch_objects_cache, 496

objvp
ch_objects_cache, 496

oc_flags_t
Objects Caches, 444

oc_hash_header_t
Objects Caches, 444

oc_lru_header_t
Objects Caches, 444

oc_object_t
Objects Caches, 444

oc_readf_t
Objects Caches, 445

oc_writef_t
Objects Caches, 445

off_ctx
chdebug_t, 539

off_flags
chdebug_t, 540

off_name
chdebug_t, 540

off_newer
chdebug_t, 540

off_older
chdebug_t, 540

off_preempt
chdebug_t, 541

off_prio
chdebug_t, 539

off_refs
chdebug_t, 541

off_state
chdebug_t, 540

off_stklimit
chdebug_t, 540

off_time
chdebug_t, 541

offset
tm_calibration_t, 567
CH_DBG_ENABLE_ASSERTS, 46
CH_DBG_ENABLE_CHECKS, 46
CH_DBG_ENABLE_STACK_CHECK, 47
CH_DBG_FILL_THREADS, 47
CH_DBG_STATISTICS, 45
CH_DBG_SYSTEM_STATE_CHECK, 45
CH_DBG_THREADS_PROFILING, 47
CH_DBG_TRACE_BUFFER_SIZE, 46
CH_DBG_TRACE_HOOK, 51
CH_DBG_TRACE_MASK, 46
CH_DBG_SYSTEM_STATE_CHECK, 45
CH_DBG_THREADS_PROFILING, 47
CH_DBG_TRACE_BUFFER_SIZE, 46
OS Library, 312

owner
ch_mutex, 493

pages
heap_header, 548
panic_msg
ch_system_debug, 517
par
ch_virtual_timer, 535
pipe
ch_dyn_pipe, 487
PIPE_DECL

Pipes, 347
pipe_list
ch_objects_factory, 499
pipe_read
Pipes, 348
pipe_t, 558
buffer, 560
cmtx, 561
cnt, 561
rdptr, 560
reset, 561
rmtx, 562
rtr, 561
top, 560
wmux, 561
wrptr, 560
wtr, 561
pipe_write
Pipes, 347
Pipes, 346

PIPE_DECL, 347
chPipeGetFreeCount, 352
chPipeGetSize, 351
chPipeGetUsedCount, 351
chPipeObjectInit, 349
chPipeReadTimeout, 350
chPipeReset, 349
chPipeResume, 352
chPipeWriteTimeout, 349

pool
guarded_memory_pool_t, 547
pool_header, 562
next, 563
pqueue
ch_thread, 520
INDEX 655

chSchDoReschedule, 110
chSchDoRescheduleAhead, 109
chSchDoRescheduleBehind, 109
chSchDoYieldS, 112
chSchGoSleepS, 105
chSchGoSleepTimeoutS, 106
chSchIsPreemptionRequired, 108
chSchIsRescRequiredI, 111
chSchPreemption, 113
chSchReadyAheadI, 104
chSchReadyI, 103
chSchRescheduleS, 108
chSchWakeupS, 107
currp, 100
delta_list_t, 101
firstprio, 100
HIGHPRIO, 96
IDLEPRIO, 95
LOWPRIO, 95
MSG_OK, 95
MSG_RESET, 95
MSG_TIMEOUT, 95
NOPRIO, 95
NORMALPRIO, 96
ready_list_t, 101
system_debug_t, 102
thread_reference_t, 101
threads_list_t, 101
threads_queue_t, 101
virtual_timer_t, 102
virtual_timers_list_t, 102
vtfunc_t, 102

sem
ch_dyn_semaphore, 488
guarded_memory_pool_t, 547

sem_list
ch_objects_factory, 499

sem_pool
ch_objects_factory, 499

SEMAPHORE_DECL
Counting Semaphores, 200

semaphore_t
Counting Semaphores, 200

sentmsg
ch_thread, 523

size
ch_trace_buffer_t, 529
chdebug_t, 538
heap_header, 548

state
ch_thread, 521
ch_trace_event_t, 531

Statistics, 308
_stats_cctxswc, 309
_stats_increase_irq, 308
_stats_init, 308
_stats_start_measure_crit_isr, 310

stats
ch_thread, 525
suspended
ch_trace_buffer_t, 529

sw
ch_trace_event_t, 532
Synchronization, 197, 316
Synchronous Messages, 267
chMsgGet, 276
chMsgIsPendingI, 275
chMsgPoll, 274
chMsgPollIS, 270
chMsgRelease, 271
chMsgReleaseS, 277
chMsgSend, 268
chMsgWait, 272
chMsgWaitS, 268
chMsgWaitTimeout, 273
chMsgWaitTimeoutS, 269

sysinterval_t
Time and Intervals, 121
System Management, 58

_idle_thread, 67
CH_FAST_IRQ_HANDLER, 62
CH_IRQ_EPILOGUE, 61
CH_IRQ_HANDLER, 61
CH_IRQ_IS_VALID_KERNEL_PRIORITY, 60
CH_IRQ_IS_VALID_PRIORITY, 60
CH_IRQ_PROLOGUE, 61
chSysDisable, 74
chSysEnable, 75
chSysGetIdleThreadX, 79
chSysGetRealtimeCounterX, 66
chSysGetStatusAndLockX, 71
chSysHalt, 68
chSysInit, 67
chSysIntegrityCheckI, 69
chSysIsCounterWithinX, 73
chSysLock, 76
chSysLockFromISR, 77
chSysPolledDelayX, 74
chSysRestoreStatusX, 72
chSysSuspend, 75
chSysSwitch, 66
chSysTimerHandlerI, 70
chSysUnconditionalLock, 78
chSysUnconditionalUnlock, 79
chSysUnlock, 76
chSysUnlockFromISR, 77
MS2RTC, 63
RTC2MS, 65
RTC2S, 64
RTC2US, 65
S2RTC, 62
THD_WORKING_AREA, 67
US2RTC, 63
system_debug_t
Scheduler, 102
systime
  ch_virtual_timers_list, 536
systime_t
  Time and Intervals, 120
tfunc_t
  Threads, 159
THD_FUNCTION
  Threads, 158
THD_WORKING_AREA
  System Management, 67
  Threads, 157
THD_WORKING_AREA_BASE
  Threads, 157
THD_WORKING_AREA_END
  Threads, 157
THD_WORKING_AREA_SIZE
  Threads, 156
thread_descriptor_t, 563
  arg, 564
  funcp, 564
  name, 564
  prio, 564
  wbase, 564
  wend, 564
thread_reference_t
  Scheduler, 101
thread_t
  Scheduler, 101
  Threads, 153
_THREADS_QUEUE_DATA, 156
_thread_init, 160
_thread_memfill, 161
chThdAddRef, 168
chThdCreate, 164
chThdCreatel, 163
chThdCreateStatic, 166
chThdCreateSuspended, 162
chThdCreateSuspendedl, 161
chThdDequeueAll, 185
chThdDequeueNextl, 184
chThdDoDequeueNextl, 191
chThdEnqueueTimeoutS, 183
chThdExit, 170
chThdExitS, 171
chThdGetPriorityX, 186
chThdGetSelfX, 186
chThdGetTicksX, 186
chThdGetWorkingAreaX, 187
chThdQueuesEmptyl, 190
chThdQueueObjectInit, 190
chThdRelease, 169
chThdResume, 182
chThdResume1, 181
chThdResumeS, 181
chThdSetPriority, 173
chThdShouldTerminateX, 188
chThdSleep, 175
chThdSleepMicroseconds, 159
chThdSleepMilliseconds, 158
chThdSleepS, 189
chThdSleepSeconds, 158
chThdSleepUntil, 176
chThdSleepUntilWindowed, 177
chThdStart, 167
chThdStartl, 188
chThdSuspendS, 179
chThdSuspendTimeoutS, 180
chThdT erminate, 174
chThdTerminatedX, 187
chThdWait, 172
chThdYield, 178
tfunc_t, 159
THD_FUNCTION, 158
THD_WORKING_AREA, 157
THD_WORKING_AREA_BASE, 157
THD_WORKING_AREA_END, 157
THD_WORKING_AREA_SIZE, 156
THREADS_QUEUE_DECL, 156
threads_list_t
  Scheduler, 101
THREADS_QUEUE_DECL
  Threads, 156
threads_queue_t
  Scheduler, 101
threadsize
chdebug_t, 539
ticks
ch_thread, 522
time
  ch_thread, 522
ch_trace_event_t, 532
Time and Intervals, 115
chTimeAddX, 125
chTimeDiffX, 126
chTimeI2MS, 124
chTimeI2S, 124
chTimeI2US, 123
chTimeIsInRangeX, 126
chTimeMS2I, 123
chTimeS2I, 122
chTimeUS2I, 123
sysinterval_t, 121
systime_t, 120
time_conv_t, 122
  TIME_I2MS, 119
  TIME_I2S, 118
  TIME_I2US, 120
  TIME_IMMEDIATE, 116
  TIME_INFINITE, 116
  TIME_MAX_INTERVAL, 116
  TIME_MAX_SYSTIME, 116
  TIME_MS2I, 117
time_msecs_t, 121
INDEX

TIME_S2I, 117
 time_secs_t, 121
TIME_US2I, 118
 time_usecs_t, 121
Time Measurement, 193
 _tm_init, 194
 chTMChainMeasurementToX, 195
 chTMOBJECTInit, 194
 chTMSStartMeasurementX, 194
 chTMStopMeasurementX, 195
 TM_CALIBRATION_LOOP, 193
time_conv_t
 Time and Intervals, 122
TIME_I2MS
 Time and Intervals, 119
TIME_I2S
 Time and Intervals, 118
TIME_I2US
 Time and Intervals, 120
TIME_IMMEDIATE
 Time and Intervals, 116
TIME_INFINITE
 Time and Intervals, 116
TIME_MAX_INTERVAL
 Time and Intervals, 116
TIME_MAX_SYSTIME
 Time and Intervals, 116
time_measurement_t, 565
 best, 566
cumulative, 566
 last, 566
 n, 566
 worst, 566
TIME_MS2I
 Time and Intervals, 117
 time_msecs_t
 Time and Intervals, 121
TIME_S2I
 Time and Intervals, 117
time_secs_t
 Time and Intervals, 121
TIME_US2I
 Time and Intervals, 118
time_usecs_t
 Time and Intervals, 121
timesize
chdebug_t, 539
tm
 ch_system, 514
TM_CALIBRATION_LOOP
 Time Measurement, 193
tm_calibration_t, 567
 offset, 567
top
 mailbox_t, 552
 pipe_t, 560
topmem
 memcore_t, 554
trace_buffer
 ch_system_debug, 517
trace_next
 Tracing, 301
Tracing, 299
 _trace_halt, 302
 _trace_init, 301
 _trace_isr_enter, 302
 _trace_isr_leave, 302
 _trace_switch, 301
 CH_DBG_TRACE_BUFFER_SIZE, 300
 CH_DBG_TRACE_MASK, 300
chDbgResumeTrace, 306
 chDbgResumeTraceI, 306
 chDbgSuspendTrace, 305
 chDbgSuspendTraceI, 304
 chDbgWriteTrace, 303
 chDbgWriteTraceI, 303
 trace_next, 301
 TRUE
 Version Numbers and Identification, 29
type
 ch_trace_event_t, 531
u
 ch_thread, 524
up1
 ch_trace_event_t, 533
up2
 ch_trace_event_t, 533
US2RTC
 System Management, 63
user
 ch_trace_event_t, 533
version
 chdebug_t, 539
Version Numbers and Identification, 27, 313
 _CHIBIOS_OSLIB_, 313
 _CHIBIOS_RT_, 27
 _oslib_init, 314
 CH_KERNEL_MAJOR, 28
 CH_KERNEL_MINOR, 28
 CH_KERNEL_PATCH, 28
 CH_KERNEL_STABLE, 28
 CH_KERNEL_VERSION, 28
 CH_OSLIB_MAJOR, 314
 CH_OSLIB_MINOR, 314
 CH_OSLIB_PATCH, 314
 CH_OSLIB_STABLE, 313
 CH_OSLIB_VERSION, 313
 chSysHalt, 29
 FALSE, 28
 TRUE, 29
Virtual Timers, 128
 _vt_init, 135
 chVTDoResetI, 137
 chVTDoSetI, 136
 chVTDoTickI, 139

ChibiOS/RT