1 ChibiOS/RT 1
  1.1 Copyright .................................................. 1
  1.2 Introduction ............................................... 1
  1.3 Related Documents ......................................... 1

2 Module Index 3
  2.1 Modules ................................................... 3

3 Hierarchical Index 5
  3.1 Class Hierarchy ........................................... 5

4 Data Structure Index 7
  4.1 Data Structures ........................................... 7

5 File Index 11
  5.1 File List ................................................... 11

6 Module Documentation 15
  6.1 Release and Licensing ...................................... 15
    6.1.1 Detailed Description .................................. 15
  6.2 Release Information ....................................... 16
    6.2.1 Detailed Description .................................. 16
    6.2.2 Macro Definition Documentation ....................... 16
      6.2.2.1 __CHIBIOS__ ........................................ 16
      6.2.2.2 CH_VERSION_STABLE ................................ 16
      6.2.2.3 CH_VERSION ......................................... 17
      6.2.2.4 CH_VERSION_YEAR ................................... 17
      6.2.2.5 CH_VERSION_MONTH .................................. 17
      6.2.2.6 CH_VERSION_PATCH .................................. 17
      6.2.2.7 CH_VERSION_NICKNAME .............................. 17
      6.2.2.8 CH_VERSION_DATE .................................. 17
  6.3 Customer Information .................................... 18
    6.3.1 Detailed Description .................................. 18
    6.3.2 Macro Definition Documentation ....................... 18
      6.3.2.1 CH_CUSTOMER_ID_STRING ............................ 19
      6.3.2.2 CH_CUSTOMER_ID_CODE ................................ 19
      6.3.2.3 CH_CUSTOMER_LICENSE_EOS_DATE .................... 19
      6.3.2.4 CH_CUSTOMER_LICENSE_VERSION_YEAR ................ 19
      6.3.2.5 CH_CUSTOMER_LICENSE_VERSION_MONTH ............... 19
      6.3.2.6 CH_LICENSE ......................................... 19
      6.3.2.7 CH_CUSTOMER_LICENSE_VERSION_DATE ................ 19
  6.4 License Settings ......................................... 20
    6.4.1 Detailed Description .................................. 20
    6.4.2 Macro Definition Documentation ....................... 20
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.8.2.19 CH_CFG_USE_CONDVARS_TIMEOUT</td>
<td>34</td>
</tr>
<tr>
<td>6.8.2.20 CH_CFG_USE_EVENTS</td>
<td>34</td>
</tr>
<tr>
<td>6.8.2.21 CH_CFG_USE_EVENTS_TIMEOUT</td>
<td>35</td>
</tr>
<tr>
<td>6.8.2.22 CH_CFG_USE_MESSAGES</td>
<td>35</td>
</tr>
<tr>
<td>6.8.2.23 CH_CFG_USE_MESSAGES_PRIORITY</td>
<td>35</td>
</tr>
<tr>
<td>6.8.2.24 CH_CFG_USE_DYNAMIC</td>
<td>36</td>
</tr>
<tr>
<td>6.8.2.25 CH_CFG_USE_MAILBOXES</td>
<td>36</td>
</tr>
<tr>
<td>6.8.2.26 CH_CFG_USE_MEMCORE</td>
<td>36</td>
</tr>
<tr>
<td>6.8.2.27 CH_CFG_MEMCORE_SIZE</td>
<td>37</td>
</tr>
<tr>
<td>6.8.2.28 CH_CFG_USE_HEAP</td>
<td>37</td>
</tr>
<tr>
<td>6.8.2.29 CH_CFG_USE_MEMPOOLS</td>
<td>37</td>
</tr>
<tr>
<td>6.8.2.30 CH_CFG_USE_OBJ_FIFOS</td>
<td>38</td>
</tr>
<tr>
<td>6.8.2.31 CH_CFG_USE_PIPES</td>
<td>38</td>
</tr>
<tr>
<td>6.8.2.32 CH_CFG_USE_OBJ_CACHES</td>
<td>38</td>
</tr>
<tr>
<td>6.8.2.33 CH_CFG_USE_DELEGATES</td>
<td>38</td>
</tr>
<tr>
<td>6.8.2.34 CH_CFG_USE_JOBS</td>
<td>39</td>
</tr>
<tr>
<td>6.8.2.35 CH_CFG_USE_FACTORY</td>
<td>39</td>
</tr>
<tr>
<td>6.8.2.36 CH_CFG_FACTORY_MAX_NAMES_LENGTH</td>
<td>39</td>
</tr>
<tr>
<td>6.8.2.37 CH_CFG_FACTORY_OBJECTS_REGISTRY</td>
<td>39</td>
</tr>
<tr>
<td>6.8.2.38 CH_CFG_FACTORY_GENERIC_BUFFERS</td>
<td>39</td>
</tr>
<tr>
<td>6.8.2.39 CH_CFG_FACTORY_SEMAPHORES</td>
<td>40</td>
</tr>
<tr>
<td>6.8.2.40 CH_CFG_FACTORY_MAILBOXES</td>
<td>40</td>
</tr>
<tr>
<td>6.8.2.41 CH_CFG_FACTORY_OBJ_FIFOS</td>
<td>40</td>
</tr>
<tr>
<td>6.8.2.42 CH_CFG_FACTORY_PIPES</td>
<td>40</td>
</tr>
<tr>
<td>6.8.2.43 CH_DBG_STATISTICS</td>
<td>40</td>
</tr>
<tr>
<td>6.8.2.44 CH_DBG_SYSTEM_STATE_CHECK</td>
<td>41</td>
</tr>
<tr>
<td>6.8.2.45 CH_DBG_ENABLE_CHECKS</td>
<td>41</td>
</tr>
<tr>
<td>6.8.2.46 CH_DBG_ENABLE_ASSERTS</td>
<td>41</td>
</tr>
<tr>
<td>6.8.2.47 CH_DBG_TRACE_MASK</td>
<td>41</td>
</tr>
<tr>
<td>6.8.2.48 CH_DBG_TRACE_BUFFER_SIZE</td>
<td>42</td>
</tr>
<tr>
<td>6.8.2.49 CH_DBG_ENABLE_STACK_CHECK</td>
<td>42</td>
</tr>
<tr>
<td>6.8.2.50 CH_DBG_FILL_THREADS</td>
<td>42</td>
</tr>
<tr>
<td>6.8.2.51 CH_DBG_THREADS_PROFILING</td>
<td>43</td>
</tr>
<tr>
<td>6.8.2.52 CH_CFG_SYSTEM_EXTRA_FIELDS</td>
<td>43</td>
</tr>
<tr>
<td>6.8.2.53 CH_CFG_SYSTEM_INIT_HOOK</td>
<td>43</td>
</tr>
<tr>
<td>6.8.2.54 CH_CFG_OS_INSTANCE_EXTRA_FIELDS</td>
<td>43</td>
</tr>
<tr>
<td>6.8.2.55 CH_CFG_OS_INSTANCE_INIT_HOOK</td>
<td>43</td>
</tr>
<tr>
<td>6.8.2.56 CH_CFG_THREAD_EXTRA_FIELDS</td>
<td>44</td>
</tr>
<tr>
<td>6.8.2.57 CH_CFG_THREAD_INIT_HOOK</td>
<td>44</td>
</tr>
<tr>
<td>6.8.2.58 CH_CFG_THREAD_EXIT_HOOK</td>
<td>44</td>
</tr>
<tr>
<td>6.8.2.59 CH_CFG_CONTEXT_SWITCH_HOOK</td>
<td>45</td>
</tr>
<tr>
<td>6.8.2.60 CH_CFG_IRQ_PROLOGUE_HOOK</td>
<td>45</td>
</tr>
</tbody>
</table>
6.8.2.61 CH_CFG_IRQ_EPILOGUE_HOOK ........................................... 45
6.8.2.62 CH_CFG_IDLE_ENTER_HOOK ........................................... 46
6.8.2.63 CH_CFG_IDLE_LEAVE_HOOK ........................................... 46
6.8.2.64 CH_CFG_IDLE_LOOP_HOOK ........................................... 46
6.8.2.65 CH_CFG_SYSTEM_TICK_HOOK ........................................... 47
6.8.2.66 CH_CFG_SYSTEM_HALT_HOOK ........................................... 47
6.8.2.67 CH_CFG_TRACE_HOOK ........................................... 47
6.8.2.68 CH_CFG_RUNTIME_FAULTS_HOOK ........................................... 47
6.9 Checks ................................................................. 48
6.10 Restrictions .......................................................... 49
  6.10.1 Detailed Description ................................................. 49
6.11 System ................................................................. 50
  6.11.1 Detailed Description ................................................. 50
6.12 Port Interface .......................................................... 51
  6.12.1 Detailed Description ................................................. 51
6.13 OS Types and Structures .............................................. 52
  6.13.1 Detailed Description ................................................. 52
  6.13.2 Macro Definition Documentation ................................... 53
    6.13.2.1 __CH_STRINGIFY ................................................. 54
    6.13.2.2 __CH_OFFSETOF ................................................. 55
    6.13.2.3 __CH_USED ................................................. 55
    6.13.2.4 likely ..................................................... 55
    6.13.2.5 unlikely ..................................................... 56
    6.13.2.6 threadref .................................................... 56
  6.13.3 Typedef Documentation ............................................. 57
    6.13.3.1 rtcnt_t ..................................................... 57
    6.13.3.2 rttime_t ..................................................... 57
    6.13.3.3 syssts_t ..................................................... 57
    6.13.3.4 stkalign_t ..................................................... 57
    6.13.3.5 tmode_t ..................................................... 57
    6.13.3.6 tstate_t ..................................................... 57
    6.13.3.7 trefs_t ..................................................... 58
    6.13.3.8 tslices_t ..................................................... 58
    6.13.3.9 tprio_t ..................................................... 58
    6.13.3.10 msg_t ..................................................... 58
    6.13.3.11 eventid_t ..................................................... 58
    6.13.3.12 eventmask_t ..................................................... 58
    6.13.3.13 eventflags_t ..................................................... 58
    6.13.3.14 cnt_t ..................................................... 58
    6.13.3.15 ucnt_t ..................................................... 58
    6.13.3.16 core_id_t ..................................................... 58
    6.13.3.17 thread_t ..................................................... 58

ChibiOS/RT
6.16.3.4 ch_delta_list_t ................................................................. 72
6.16.4 Function Documentation ...................................................... 72
  6.16.4.1 ch_list_init() .............................................................. 72
  6.16.4.2 ch_list_isempty() .......................................................... 72
  6.16.4.3 ch_list_notempty() .......................................................... 73
  6.16.4.4 ch_list_link() ............................................................... 73
  6.16.4.5 ch_list_unlink() ............................................................ 73
  6.16.4.6 ch_queue_init() ............................................................. 74
  6.16.4.7 ch_queue_isempty() ......................................................... 74
  6.16.4.8 ch_queue_notempty() ....................................................... 75
  6.16.4.9 ch_queue_insert() .......................................................... 75
  6.16.4.10 ch_queue_fifo_remove() .................................................. 75
  6.16.4.11 ch_queue_lifo_remove() .................................................. 76
  6.16.4.12 ch_queue_dequeue() ....................................................... 77
  6.16.4.13 ch_pqueue_init() .......................................................... 77
  6.16.4.14 ch_pqueue_remove_highest() ............................................. 77
  6.16.4.15 ch_pqueue_insert_behind() .............................................. 78
  6.16.4.16 ch_pqueue_insert_ahead() ............................................... 78
  6.16.4.17 ch_dlist_init() ............................................................ 79
  6.16.4.18 ch_dlist_isempty() ........................................................ 79
  6.16.4.19 ch_dlist_notempty() ...................................................... 80
  6.16.4.20 ch_dlist_islast() .......................................................... 80
  6.16.4.21 ch_dlist_isfirst() ........................................................ 80
  6.16.4.22 ch_dlist_insert_after() ................................................ 81
  6.16.4.23 ch_dlist_insert_before() ............................................... 81
  6.16.4.24 ch_dlist_insert() ........................................................ 82
  6.16.4.25 ch_dlist_remove_first() ................................................. 82
  6.16.4.26 ch_dlist_dequeue() ....................................................... 83
6.17 Scheduler ............................................................................. 84
  6.17.1 Detailed Description .......................................................... 84
  6.17.2 Macro Definition Documentation .......................................... 87
    6.17.2.1 MSG_OK .................................................................. 87
    6.17.2.2 MSG_TIMEOUT .............................................................. 87
    6.17.2.3 MSG_RESET ................................................................. 87
    6.17.2.4 NOPRIO .................................................................. 87
    6.17.2.5 IDLEPRIO ................................................................. 87
    6.17.2.6 LOWPRIO ................................................................. 88
    6.17.2.7 NORMALPRIO .............................................................. 88
    6.17.2.8 HIGHPRIO ................................................................. 88
    6.17.2.9 CH_STATE_READY ......................................................... 88
    6.17.2.10 CH_STATE_CURRENT .................................................... 88
    6.17.2.11 CH_STATE_WTSTART ................................................... 88
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.19.2.1</td>
<td>CH_SYS_CORE0_MEMORY</td>
<td>108</td>
</tr>
<tr>
<td>6.19.2.2</td>
<td>CH_SYS_CORE1_MEMORY</td>
<td>109</td>
</tr>
<tr>
<td>6.19.2.3</td>
<td>currcore</td>
<td>109</td>
</tr>
<tr>
<td>6.19.2.4</td>
<td>CH_IRQ_IS_VALID_PRIORITY</td>
<td>109</td>
</tr>
<tr>
<td>6.19.2.5</td>
<td>CH_IRQ_IS_VALID_KERNEL_PRIORITY</td>
<td>109</td>
</tr>
<tr>
<td>6.19.2.6</td>
<td>CH_IRQ_PROLOGUE</td>
<td>110</td>
</tr>
<tr>
<td>6.19.2.7</td>
<td>CH_IRQ_EPILOGUE</td>
<td>111</td>
</tr>
<tr>
<td>6.19.2.8</td>
<td>CH_IRQ_HANDLER</td>
<td>111</td>
</tr>
<tr>
<td>6.19.2.9</td>
<td>CH_FAST_IRQHandler</td>
<td>111</td>
</tr>
<tr>
<td>6.19.2.10</td>
<td>S2RTC</td>
<td>112</td>
</tr>
<tr>
<td>6.19.2.11</td>
<td>MS2RTC</td>
<td>112</td>
</tr>
<tr>
<td>6.19.2.12</td>
<td>US2RTC</td>
<td>113</td>
</tr>
<tr>
<td>6.19.2.13</td>
<td>RTC2S</td>
<td>113</td>
</tr>
<tr>
<td>6.19.2.14</td>
<td>RTC2MS</td>
<td>114</td>
</tr>
<tr>
<td>6.19.2.15</td>
<td>RTC2US</td>
<td>115</td>
</tr>
<tr>
<td>6.19.2.16</td>
<td>chSysGetRealtimeCounterX</td>
<td>115</td>
</tr>
<tr>
<td>6.19.2.17</td>
<td>chSysSwitch</td>
<td>116</td>
</tr>
<tr>
<td>6.19.3</td>
<td>Function Documentation</td>
<td>116</td>
</tr>
<tr>
<td>6.19.3.1</td>
<td>THD_WORKING_AREA()</td>
<td>116</td>
</tr>
<tr>
<td>6.19.3.2</td>
<td>THD_WORKING_AREA()</td>
<td>116</td>
</tr>
<tr>
<td>6.19.3.3</td>
<td>chSysWaitSystemState()</td>
<td>117</td>
</tr>
<tr>
<td>6.19.3.4</td>
<td>chSysInit()</td>
<td>117</td>
</tr>
<tr>
<td>6.19.3.5</td>
<td>chSysHalt()</td>
<td>117</td>
</tr>
<tr>
<td>6.19.3.6</td>
<td>chSysGetIdleThreadX()</td>
<td>118</td>
</tr>
<tr>
<td>6.19.3.7</td>
<td>chSysIntegrityCheck()</td>
<td>119</td>
</tr>
<tr>
<td>6.19.3.8</td>
<td>chSysTimerHandler()</td>
<td>120</td>
</tr>
<tr>
<td>6.19.3.9</td>
<td>chSysGetStatusAndLockX()</td>
<td>121</td>
</tr>
<tr>
<td>6.19.3.10</td>
<td>chSysRestoreStatusX()</td>
<td>121</td>
</tr>
<tr>
<td>6.19.3.11</td>
<td>chSysIsCounterWithinX()</td>
<td>122</td>
</tr>
<tr>
<td>6.19.3.12</td>
<td>chSysPolledDelayX()</td>
<td>123</td>
</tr>
<tr>
<td>6.19.3.13</td>
<td>chSysDisable()</td>
<td>124</td>
</tr>
<tr>
<td>6.19.3.14</td>
<td>chSysSuspend()</td>
<td>124</td>
</tr>
<tr>
<td>6.19.3.15</td>
<td>chSysEnable()</td>
<td>125</td>
</tr>
<tr>
<td>6.19.3.16</td>
<td>chSysLock()</td>
<td>126</td>
</tr>
<tr>
<td>6.19.3.17</td>
<td>chSysUnlock()</td>
<td>126</td>
</tr>
<tr>
<td>6.19.3.18</td>
<td>chSysLockFromISR()</td>
<td>127</td>
</tr>
<tr>
<td>6.19.3.19</td>
<td>chSysUnlockFromISR()</td>
<td>127</td>
</tr>
<tr>
<td>6.19.3.20</td>
<td>chSysUnconditionalLock()</td>
<td>128</td>
</tr>
<tr>
<td>6.19.3.21</td>
<td>chSysUnconditionalUnlock()</td>
<td>129</td>
</tr>
<tr>
<td>6.19.3.22</td>
<td>chSysNotifyInstance()</td>
<td>129</td>
</tr>
<tr>
<td>6.19.4</td>
<td>Variable Documentation</td>
<td>129</td>
</tr>
<tr>
<td>6.19.4.1</td>
<td>ch_system</td>
<td>130</td>
</tr>
</tbody>
</table>
6.19.4.2 ch0 ................................................................. 130
6.19.4.3 ch_core0_cfg .............................................. 130
6.19.4.4 ch1 ................................................................. 130
6.19.4.5 ch_core1_cfg .............................................. 130
6.20 Memory Alignment .................................................. 131
6.20.1 Detailed Description ............................................. 131
6.20.2 Macro Definition Documentation ................................. 131
   6.20.2.1 MEM_ALIGN_MASK ......................................... 131
   6.20.2.2 MEM_ALIGN_PREV ......................................... 131
   6.20.2.3 MEM_ALIGN_NEXT ......................................... 132
   6.20.2.4 MEM_ISAligned ........................................... 132
   6.20.2.5 MEM_IS_VALID_ALIGNMENT ................................. 132
6.21 Time and Intervals ................................................. 134
6.21.1 Detailed Description ............................................. 134
6.21.2 Macro Definition Documentation ................................. 135
   6.21.2.1 TIME_IMMEDIATE .......................................... 135
   6.21.2.2 TIME_INFINITY ........................................... 136
   6.21.2.3 TIME_MAX_INTERVAL ...................................... 136
   6.21.2.4 TIME_MAX_SYSTIME ....................................... 136
   6.21.2.5 TIME_S2I .................................................. 136
   6.21.2.6 TIME_MS2I ............................................... 137
   6.21.2.7 TIME_US2I ................................................ 137
   6.21.2.8 TIME_I2S .................................................. 138
   6.21.2.9 TIME_I2MS ................................................. 139
   6.21.2.10 TIME_I2US ............................................... 139
6.21.3 Typedef Documentation .......................................... 140
   6.21.3.1 systime_t ............................................... 140
   6.21.3.2 sysinterval_t ............................................ 140
   6.21.3.3 systimestamp_t .......................................... 140
   6.21.3.4 time_secs_t ............................................. 141
   6.21.3.5 time_msecs_t ........................................... 141
   6.21.3.6 time_usecs_t ........................................... 141
   6.21.3.7 time_conv_t ............................................ 141
6.21.4 Function Documentation ......................................... 141
   6.21.4.1 chTimeS2I() ............................................. 141
   6.21.4.2 chTimeMS2I() ............................................ 142
   6.21.4.3 chTimeUS2I() ............................................ 142
   6.21.4.4 chTimeI2S() ............................................. 143
   6.21.4.5 chTimeI2MS() ............................................ 143
   6.21.4.6 chTimeI2US() ............................................ 144
   6.21.4.7 chTimeAddX() ............................................ 144
   6.21.4.8 chTimeDiffX() ............................................ 145
6.21.4.9 chTimelsInRangeX()  .......................................................... 145
6.21.4.10 chTimeStamPAddX() ......................................................... 146
6.21.4.11 chTimeStamPDiffX() ......................................................... 146
6.21.4.12 chTimeStampsInRangeX() .................................................. 147
6.22 Virtual Timers ........................................................................ 148
6.22.1 Detailed Description ............................................................. 148
6.22.2 Function Documentation ......................................................... 149
6.22.2.1 vt_set_alarm() ................................................................. 149
6.22.2.2 vt_insert_first() .............................................................. 150
6.22.2.3 vt_enqueue() ................................................................. 151
6.22.2.4 chVTDoSetI() ................................................................. 152
6.22.2.5 chVTDoSetContinuousI() .................................................. 153
6.22.2.6 chVTDoResetI() .............................................................. 154
6.22.2.7 chVTGetRemainingIntervalI() ........................................... 155
6.22.2.8 chVTDoTickI() ............................................................... 156
6.22.2.9 chVTGetTimeStampI() ...................................................... 157
6.22.2.10 chVTResetTimeStampI() .................................................. 158
6.22.2.11 chVTObjectInit() ............................................................ 159
6.22.2.12 chVTGetSystemTimeX() .................................................. 159
6.22.2.13 chVTGetSystemTime() ...................................................... 160
6.22.2.14 chVTTimeElapsedSinceX() .............................................. 160
6.22.2.15 chVTIsSystemTimeWithinX() .......................................... 161
6.22.2.16 chVTIsSystemTimeWithin() ............................................. 162
6.22.2.17 chVTGetTimersStateI() ................................................... 163
6.22.2.18 chVTIsArmedI() ............................................................ 164
6.22.2.19 chVTIsArmed() ............................................................. 165
6.22.2.20 chVTResetI() ............................................................... 166
6.22.2.21 chVTReset() ................................................................. 167
6.22.2.22 chVTSetI() ................................................................. 168
6.22.2.23 chVTSet() ................................................................. 169
6.22.2.24 chVTSetContinuousI() .................................................. 170
6.22.2.25 chVTSetContinuous() ..................................................... 171
6.22.2.26 chVTGetReloadIntervalX() ............................................. 172
6.22.2.27 chVTSetReloadIntervalX() ............................................. 173
6.22.2.28 chVTGetTimeStamp() .................................................... 173
6.22.2.29 chVTResetTimeStamp() .................................................. 174
6.22.2.30 __vt_object_init() ......................................................... 175
6.23 Threads .................................................................................. 177
6.23.1 Detailed Description ............................................................. 177
6.23.2 Macro Definition Documentation ........................................... 180
6.23.2.1 __THREADS_QUEUE_DATA .............................................. 180
6.23.2.2 THREADS_QUEUE_DECL .................................................. 180

ChibiOS/RT
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.26.4.14</td>
<td>chSemFastWaitI()</td>
<td>240</td>
</tr>
<tr>
<td>6.26.4.15</td>
<td>chSemFastSignalI()</td>
<td>241</td>
</tr>
<tr>
<td>6.26.4.16</td>
<td>chSemGetCounterI()</td>
<td>241</td>
</tr>
<tr>
<td>6.27</td>
<td>Mutexes</td>
<td>243</td>
</tr>
<tr>
<td>6.27.1</td>
<td>Detailed Description</td>
<td>243</td>
</tr>
<tr>
<td>6.27.2</td>
<td>Macro Definition Documentation</td>
<td>244</td>
</tr>
<tr>
<td>6.27.2.1</td>
<td>__MUTEX_DATA</td>
<td>245</td>
</tr>
<tr>
<td>6.27.2.2</td>
<td>MUTEX_DECL</td>
<td>246</td>
</tr>
<tr>
<td>6.27.3</td>
<td>Typedef Documentation</td>
<td>246</td>
</tr>
<tr>
<td>6.27.3.1</td>
<td>mutex_t</td>
<td>246</td>
</tr>
<tr>
<td>6.27.4</td>
<td>Function Documentation</td>
<td>246</td>
</tr>
<tr>
<td>6.27.4.1</td>
<td>chMtxObjectInit()</td>
<td>246</td>
</tr>
<tr>
<td>6.27.4.2</td>
<td>chMtxLock()</td>
<td>247</td>
</tr>
<tr>
<td>6.27.4.3</td>
<td>chMtxLockS()</td>
<td>248</td>
</tr>
<tr>
<td>6.27.4.4</td>
<td>chMtxTryLock()</td>
<td>249</td>
</tr>
<tr>
<td>6.27.4.5</td>
<td>chMtxTryLockS()</td>
<td>250</td>
</tr>
<tr>
<td>6.27.4.6</td>
<td>chMtxUnlock()</td>
<td>251</td>
</tr>
<tr>
<td>6.27.4.7</td>
<td>chMtxUnlockS()</td>
<td>252</td>
</tr>
<tr>
<td>6.27.4.8</td>
<td>chMtxUnlockAll()</td>
<td>253</td>
</tr>
<tr>
<td>6.27.4.9</td>
<td>chMtxUnlockAllS()</td>
<td>254</td>
</tr>
<tr>
<td>6.27.4.10</td>
<td>chMtxQueueNotEmptyS()</td>
<td>255</td>
</tr>
<tr>
<td>6.27.4.11</td>
<td>chMtxGetOwnerI()</td>
<td>256</td>
</tr>
<tr>
<td>6.27.4.12</td>
<td>chMtxGetNextMutexX()</td>
<td>257</td>
</tr>
<tr>
<td>6.28</td>
<td>Condition Variables</td>
<td>258</td>
</tr>
<tr>
<td>6.28.1</td>
<td>Detailed Description</td>
<td>258</td>
</tr>
<tr>
<td>6.28.2</td>
<td>Macro Definition Documentation</td>
<td>259</td>
</tr>
<tr>
<td>6.28.2.1</td>
<td>__CONDVAR_DATA</td>
<td>259</td>
</tr>
<tr>
<td>6.28.2.2</td>
<td>CONDVARIABLE_DECL</td>
<td>259</td>
</tr>
<tr>
<td>6.28.3</td>
<td>Typedef Documentation</td>
<td>259</td>
</tr>
<tr>
<td>6.28.3.1</td>
<td>condition_variable_t</td>
<td>259</td>
</tr>
<tr>
<td>6.28.4</td>
<td>Function Documentation</td>
<td>259</td>
</tr>
<tr>
<td>6.28.4.1</td>
<td>chCondObjectInit()</td>
<td>260</td>
</tr>
<tr>
<td>6.28.4.2</td>
<td>chCondSignal()</td>
<td>260</td>
</tr>
<tr>
<td>6.28.4.3</td>
<td>chCondSignalI()</td>
<td>261</td>
</tr>
<tr>
<td>6.28.4.4</td>
<td>chCondBroadcast()</td>
<td>262</td>
</tr>
<tr>
<td>6.28.4.5</td>
<td>chCondBroadcastI()</td>
<td>263</td>
</tr>
<tr>
<td>6.28.4.6</td>
<td>chCondWait()</td>
<td>264</td>
</tr>
<tr>
<td>6.28.4.7</td>
<td>chCondWaitS()</td>
<td>265</td>
</tr>
<tr>
<td>6.28.4.8</td>
<td>chCondWaitTimeout()</td>
<td>266</td>
</tr>
<tr>
<td>6.28.4.9</td>
<td>chCondWaitTimeoutS()</td>
<td>268</td>
</tr>
<tr>
<td>6.29</td>
<td>Event Flags</td>
<td>270</td>
</tr>
<tr>
<td>6.29.1</td>
<td>Detailed Description</td>
<td>270</td>
</tr>
</tbody>
</table>
6.29.2 Macro Definition Documentation ............................................... 272
  6.29.2.1 ALL_EVENTS ................................................................. 272
  6.29.2.2 EVENT_MASK .............................................................. 272
  6.29.2.3 __EVENTSOURCE_DATA ................................................ 272
  6.29.2.4 EVENTSOURCE_DECL .................................................. 273
6.29.3 Typedef Documentation ........................................................... 273
  6.29.3.1 event_source_t ......................................................... 273
  6.29.3.2 evhandler_t .............................................................. 273
6.29.4 Function Documentation .......................................................... 273
  6.29.4.1 chEvtRegisterMaskWithFlagsI() .................................... 274
  6.29.4.2 chEvtRegisterMaskWithFlags() ..................................... 274
  6.29.4.3 chEvtUnregister() ..................................................... 275
  6.29.4.4 chEvtGetAndClearEventsI() ......................................... 276
  6.29.4.5 chEvtGetAndClearEvents() ......................................... 277
  6.29.4.6 chEvtAddEvents() ..................................................... 278
  6.29.4.7 chEvtGetAndClearFlagsI() ......................................... 279
  6.29.4.8 chEvtGetAndClearFlags() ......................................... 280
  6.29.4.9 chEvtSignalI() ......................................................... 281
  6.29.4.10 chEvtSignal() ........................................................ 282
  6.29.4.11 chEvtBroadcastFlagsI() ........................................... 283
  6.29.4.12 chEvtBroadcastFlags() ............................................ 284
  6.29.4.13 chEvtDispatch() ..................................................... 285
  6.29.4.14 chEvtWaitOne() ..................................................... 285
  6.29.4.15 chEvtWaitAny() ..................................................... 286
  6.29.4.16 chEvtWaitAll() ...................................................... 287
  6.29.4.17 chEvtWaitOneTimeout() ............................................ 288
  6.29.4.18 chEvtWaitAnyTimeout() ............................................ 289
  6.29.4.19 chEvtWaitAllTimeout() ............................................ 290
  6.29.4.20 chEvtObjectInit() .................................................. 291
  6.29.4.21 chEvtRegisterMask() .............................................. 292
  6.29.4.22 chEvtRegister() ..................................................... 292
  6.29.4.23 chEvtIsListeningI() .............................................. 293
  6.29.4.24 chEvtBroadcast() .................................................. 294
  6.29.4.25 chEvtBroadcastI() ................................................ 294
  6.29.4.26 chEvtAddEventsI() ................................................ 295
  6.29.4.27 chEvtGetEventsX() ............................................... 295
6.30 Synchronous Messages .............................................................. 297
  6.30.1 Detailed Description ...................................................... 297
  6.30.2 Function Documentation ................................................... 298
    6.30.2.1 chMsgSend() ....................................................... 298
    6.30.2.2 chMsgWaitS() ...................................................... 298
    6.30.2.3 chMsgWaitTimeoutS() ............................................ 299
6.34.3 __dbg_check_enable()  ........................................ 327
6.34.4 __dbg_check_lock()  .......................................... 327
6.34.5 __dbg_check_unlock() ......................................... 328
6.34.6 __dbg_check_lock_from_isr() .................................. 328
6.34.7 __dbg_check_unlock_from_isr() ................................ 329
6.34.8 __dbg_check_enter_isr() ...................................... 329
6.34.9 __dbg_check_leave_isr() ...................................... 330
6.34.10 chDbgCheckClassI() ......................................... 330
6.34.11 chDbgCheckClassS() .......................................... 331
6.34.12 __dbg_object_init() .......................................... 331

6.35 Tracing ............................................................... 332

6.35.1 Detailed Description ........................................... 332
6.35.2 Macro Definition Documentation ................................. 333
   6.35.2.1 CH_DBG_TRACE_MASK ...................................... 333
   6.35.2.2 CH_DBG_TRACE_BUFFER_SIZE ................................. 333
6.35.3 Function Documentation ........................................ 334
   6.35.3.1 trace_next() ................................................ 334
   6.35.3.2 __trace_object_init() .................................... 334
   6.35.3.3 __trace_ready() .......................................... 335
   6.35.3.4 __trace_switch() ......................................... 335
   6.35.3.5 __trace_isr_enter() ...................................... 335
   6.35.3.6 __trace_isr_leave() ...................................... 336
   6.35.3.7 __trace_halt() ........................................... 336
   6.35.3.8 chTraceWriteI() .......................................... 336
   6.35.3.9 chTraceWrite() ........................................... 337
   6.35.3.10 chTraceSuspendI() ...................................... 338
   6.35.3.11 chTraceSuspend() ........................................ 339
   6.35.3.12 chTraceResumeI() ....................................... 340
   6.35.3.13 chTraceResume() ........................................ 341

6.36 Statistics ........................................................... 342

6.36.1 Detailed Description ........................................... 342
6.36.2 Function Documentation ........................................ 342
   6.36.2.1 __stats_increase_irq() ................................ 342
   6.36.2.2 __stats_ctxswc() ........................................ 342
   6.36.2.3 __stats_start_measure_crit_thd() ....................... 343
   6.36.2.4 __stats_stop_measure_crit_thd() ....................... 343
   6.36.2.5 __stats_start_measure_crit_isr() ....................... 344
   6.36.2.6 __stats_stop_measure_crit_isr() ....................... 344
   6.36.2.7 __stats_object_init() ................................ 345

6.37 OS Library .......................................................... 346

6.37.1 Detailed Description ........................................... 346
6.38 Version Numbers and Identification ................................ 347
6.41.3.7 chMBPostAheadTimeout() .................................................. 370
6.41.3.8 chMBPostAheadTimeoutS() ................................................ 371
6.41.3.9 chMBPostAheadI() ............................................................ 372
6.41.3.10 chMBFetchTimeout() ...................................................... 373
6.41.3.11 chMBFetchTimeoutS() ..................................................... 374
6.41.3.12 chMBFetchI() ................................................................. 375
6.41.3.13 chMBGetSizeI() ............................................................... 376
6.41.3.14 chMBGetUsedCountI() .................................................... 377
6.41.3.15 chMBGetFreeCountI() ..................................................... 377
6.41.3.16 chMBPeekI() ................................................................. 378
6.41.3.17 chMBResumeX() ............................................................. 379

6.42 Pipes ................................................................. 380
6.42.1 Detailed Description .......................................................... 380
6.42.2 Macro Definition Documentation ........................................... 380
6.42.2.1 __PIPE_DATA ................................................................. 381
6.42.2.2 PIPE_DECL ................................................................. 381
6.42.3 Function Documentation ...................................................... 381
6.42.3.1 pipe_write() ................................................................. 382
6.42.3.2 pipe_read() ................................................................. 382
6.42.3.3 chPipeObjectInit() ....................................................... 383
6.42.3.4 chPipeReset() .............................................................. 383
6.42.3.5 chPipeWriteTimeout() ................................................... 384
6.42.3.6 chPipeReadTimeout() .................................................... 384
6.42.3.7 chPipeGetSize() ............................................................ 385
6.42.3.8 chPipeGetUsedCount() ................................................. 385
6.42.3.9 chPipeGetFreeCount() .................................................. 386
6.42.3.10 chPipeResume() ........................................................... 386

6.43 Delegate Threads .......................................................... 388
6.43.1 Detailed Description .......................................................... 388
6.43.2 Typedef Documentation ...................................................... 389
6.43.2.1 delegate_veneer_t .......................................................... 389
6.43.2.2 delegate_fn0_t .............................................................. 389
6.43.2.3 delegate_fn1_t .............................................................. 389
6.43.2.4 delegate_fn2_t .............................................................. 389
6.43.2.5 delegate_fn3_t .............................................................. 389
6.43.2.6 delegate_fn4_t .............................................................. 389
6.43.3 Function Documentation ...................................................... 389
6.43.3.1 __ch_delegate_fn0() ..................................................... 389
6.43.3.2 __ch_delegate_fn1() ..................................................... 390
6.43.3.3 __ch_delegate_fn2() ..................................................... 390
6.43.3.4 __ch_delegate_fn3() ..................................................... 390
6.43.3.5 __ch_delegate_fn4() ..................................................... 392
6.43.3.6 chDelegateCallVeneer() .......................... 392
6.43.3.7 chDelegateDispatch() ............................ 393
6.43.3.8 chDelegateDispatchTimeout() .................. 394
6.43.3.9 chDelegateCallDirect0() ......................... 395
6.43.3.10 chDelegateCallDirect1() ....................... 395
6.43.3.11 chDelegateCallDirect2() ....................... 396
6.43.3.12 chDelegateCallDirect3() ....................... 397
6.43.3.13 chDelegateCallDirect4() ....................... 398

6.44 Jobs Queues .............................................. 400
  6.44.1 Detailed Description ................................ 400
  6.44.2 Macro Definition Documentation .................. 401
    6.44.2.1 MSG_JOB_NULL .................................. 401
  6.44.3 Typedef Documentation ............................. 401
    6.44.3.1 jobs_queue_t ................................... 401
    6.44.3.2 job_function_t ................................ 401
    6.44.3.3 job_descriptor_t ............................... 401
  6.44.4 Function Documentation ............................ 401
    6.44.4.1 chJobObjectInit() ................................ 401
    6.44.4.2 chJobGet() ..................................... 402
    6.44.4.3 chJobGetI() .................................... 403
    6.44.4.4 chJobGetTimeoutS() ............................ 404
    6.44.4.5 chJobGetTimeout() ................................ 405
    6.44.4.6 chJobPostI() .................................... 406
    6.44.4.7 chJobPostS() .................................... 407
    6.44.4.8 chJobPost() ..................................... 408
    6.44.4.9 chJobPostAheadI() ................................ 409
    6.44.4.10 chJobPostAheadS() ................................ 410
    6.44.4.11 chJobPostAhead() ................................ 411
    6.44.4.12 chJobDispatch() .................................. 412
    6.44.4.13 chJobDispatchTimeout() ......................... 413

6.45 Memory Management ..................................... 415
  6.45.1 Detailed Description ................................ 415
  6.46 Core Memory Manager .................................. 416
    6.46.1 Detailed Description ................................ 416
    6.46.2 Macro Definition Documentation .................. 417
      6.46.2.1 CH_CFG_MEMCORE_SIZE .......................... 417
      6.46.2.2 chCoreAllocAlignedWithOffsetI .............. 418
      6.46.2.3 chCoreAllocAlignedWithOffset .............. 418
    6.46.3 Typedef Documentation ............................. 418
      6.46.3.1 memgetfunc_t .................................. 418
      6.46.3.2 memgetfunc2_t ................................ 418
    6.46.4 Function Documentation ............................ 419
6.46.4.1 __core_init() .......................................................... 419
6.46.4.2 chCoreAllocFromBase() ........................................... 419
6.46.4.3 chCoreAllocFromTop() ............................................. 420
6.46.4.4 chCoreAllocFromBase() ............................................. 421
6.46.4.5 chCoreAllocFromTop() ............................................. 422
6.46.4.6 chCoreGetStatusX() ............................................... 423
6.46.4.7 chCoreAllocAligned() ............................................. 423
6.46.4.8 chCoreAllocAligned() ............................................. 424
6.46.4.9 chCoreAlloc() .................................................... 425
6.46.4.10 chCoreAlloc() ................................................... 425

6.46.5 Variable Documentation ........................................... 426
6.46.5.1 ch_memcore ....................................................... 426

6.47 Memory Heaps ......................................................... 427
6.47.1 Detailed Description .............................................. 427
6.47.2 Macro Definition Documentation .................................. 428
6.47.2.1 CH_HEAP_ALIGNMENT ........................................... 428
6.47.2.2 CH_HEAP_AREA ................................................ 428
6.47.3 Typedef Documentation ............................................ 429
6.47.3.1 memory_heap_t .................................................. 429
6.47.3.2 heap_header_t .................................................. 429
6.47.4 Function Documentation .......................................... 429
6.47.4.1 __heap_init() ................................................... 429
6.47.4.2 chHeapObjectInit() ............................................. 429
6.47.4.3 chHeapAllocAligned() ......................................... 430
6.47.4.4 chHeapFree() .................................................. 430
6.47.4.5 chHeapStatus() ................................................. 431
6.47.4.6 chHeapAlloc() ................................................ 431
6.47.4.7 chHeapGetSize() ............................................... 432

6.47.5 Variable Documentation ........................................... 433
6.47.5.1 default_heap .................................................... 433

6.48 Memory Pools .......................................................... 434
6.48.1 Detailed Description ............................................. 434
6.48.2 Macro Definition Documentation .................................. 435
6.48.2.1 __MEMORYPOOL_DATA ........................................ 436
6.48.2.2 MEMORYPOOL_DECL ........................................ 436
6.48.2.3 __GUARDEDMEMORYPOOL_DATA .............................. 436
6.48.2.4 GUARDEDMEMORYPOOL_DECL ............................... 437
6.48.3 Function Documentation .......................................... 437
6.48.3.1 chPoolObjectInitAligned() .................................. 437
6.48.3.2 chPoolLoadArray() ............................................ 438
6.48.3.3 chPoolAlloc() ................................................ 439
6.48.3.4 chPoolAlloc() ................................................ 440
6.48.3.5 chPoolFreeI() ................................. 441
6.48.3.6 chPoolFree() ................................ 442
6.48.3.7 chGuardedPoolObjectInitAligned() .... 443
6.48.3.8 chGuardedPoolLoadArray() ................ 444
6.48.3.9 chGuardedPoolAllocTimeoutS() ...... 445
6.48.3.10 chGuardedPoolAllocTimeout() ...... 446
6.48.3.11 chGuardedPoolFree() ....................... 447
6.48.3.12 chPoolObjectInit() ......................... 448
6.48.3.13 chPoolAdd() ................................ 449
6.48.3.14 chPoolAddI() ................................ 450
6.48.3.15 chGuardedPoolObjectInit() .......... 451
6.48.3.16 chGuardedPoolGetCounterI() .......... 451
6.48.3.17 chGuardedPoolAlloc() ...................... 452
6.48.3.18 chGuardedPoolAllocI() ................... 453
6.48.3.19 chGuardedPoolAllocS() ................. 454
6.48.3.20 chGuardedPoolAdd() ....................... 455
6.48.3.21 chGuardedPoolAddI() ...................... 456
6.48.3.22 chGuardedPoolAddS() ...................... 457

6.49 Complex Services ................................ 459
6.49.1 Detailed Description ............................ 459

6.50 Objects FIFOs .................................... 460
6.50.1 Detailed Description ............................ 460
6.50.2 Typedef Documentation ........................... 461
6.50.2.1 objects_fifo_t ............................... 461
6.50.3 Function Documentation .......................... 461
6.50.3.1 chFifoObjectInitAligned() .............. 461
6.50.3.2 chFifoObjectInit() .......................... 462
6.50.3.3 chFifoTakeObjectI() ......................... 463
6.50.3.4 chFifoTakeObjectTimeoutS() ............ 464
6.50.3.5 chFifoTakeObjectTimeout() ............. 465
6.50.3.6 chFifoReturnObjectI() ..................... 466
6.50.3.7 chFifoReturnObjectS() ..................... 467
6.50.3.8 chFifoReturnObject() ....................... 467
6.50.3.9 chFifoSendObjectI() ....................... 468
6.50.3.10 chFifoSendObjectS() ...................... 469
6.50.3.11 chFifoSendObject() ....................... 470
6.50.3.12 chFifoSendObjectAheadI() ............. 471
6.50.3.13 chFifoSendObjectAheadS() ............ 472
6.50.3.14 chFifoSendObjectAhead() ........... 473
6.50.3.15 chFifoReceiveObjectI() .............. 473
6.50.3.16 chFifoReceiveObjectTimeoutS() .... 474
6.50.3.17 chFifoReceiveObjectTimeout() .. 475
6.51 Objects Caches ................................................................. 477
6.51.1 Detailed Description ...................................................... 477
6.51.2 Typedef Documentation .................................................. 478
  6.51.2.1 oc_flags_t .......................................................... 478
  6.51.2.2 oc_hash_header_t ................................................ 478
  6.51.2.3 oc_lru_header_t .................................................. 478
  6.51.2.4 oc_object_t ...................................................... 479
  6.51.2.5 objects_cache_t .................................................. 479
  6.51.2.6 oc_readf_t ........................................................ 479
  6.51.2.7 oc_writef_t ........................................................ 479
6.51.3 Function Documentation ............................................... 479
  6.51.3.1 hash_get_s() .................................................... 480
  6.51.3.2 lru_get_last_s() ............................................... 480
  6.51.3.3 chCacheObjectInit() .......................................... 481
  6.51.3.4 chCacheGetObject() .......................................... 482
  6.51.3.5 chCacheReleaseObjectI() .................................... 483
  6.51.3.6 chCacheReadObject() ......................................... 484
  6.51.3.7 chCacheWriteObject() ......................................... 484
  6.51.3.8 chCacheReleaseObject() ...................................... 485
6.52 Dynamic Objects Factory .................................................. 487
6.52.1 Detailed Description .................................................. 487
6.52.2 Macro Definition Documentation ..................................... 490
  6.52.2.1 CH_CFG_FACTORY_MAX_NAMES_LENGTH ......................... 490
  6.52.2.2 CH_CFG_FACTORY_OBJECTS_REGISTRY .............................. 490
  6.52.2.3 CH_CFG_FACTORY_GENERIC_BUFFERS .............................. 490
  6.52.2.4 CH_CFG_FACTORY_SEMAPHORES [1/2] ............................ 490
  6.52.2.5 CH_CFG_FACTORY_SEMAPHORES [2/2] ............................ 491
  6.52.2.6 CH_CFG_FACTORY_MAILBOXES [1/2] ............................. 491
  6.52.2.7 CH_CFG_FACTORY_MAILBOXES [2/2] ............................. 491
  6.52.2.8 CH_CFG_FACTORY_OBJ_FIFOS [1/3] .............................. 491
  6.52.2.9 CH_CFG_FACTORY_OBJ_FIFOS [2/3] .............................. 491
  6.52.2.10 CH_CFG_FACTORY_OBJ_FIFO [3/3] ............................ 491
  6.52.2.11 CH_CFG_FACTORY_PIPES [1/2] ................................. 492
  6.52.2.12 CH_CFG_FACTORY_PIPES [2/2] ................................. 492
6.52.3 Typedef Documentation ............................................... 492
  6.52.3.1 dyn_element_t .................................................. 492
  6.52.3.2 dyn_list_t ...................................................... 492
  6.52.3.3 registered_object_t ........................................... 492
  6.52.3.4 dyn_buffer_t .................................................... 492
  6.52.3.5 dyn_semaphore_t ............................................... 493
  6.52.3.6 dyn_mailbox_t .................................................. 493
  6.52.3.7 dyn_objects_fifo_t ............................................. 493
6.52.3.8 dyn_pipe_t ........................................ 493
6.52.3.9 objects_factory_t ................................. 493

6.52.4 Function Documentation ................................ 493
6.52.4.1 __factory_init() .................................. 494
6.52.4.2 chFactoryRegisterObject() ....................... 494
6.52.4.3 chFactoryFindObject() ............................ 495
6.52.4.4 chFactoryFindObjectByPointer() ................. 495
6.52.4.5 chFactoryReleaseObject() ....................... 496
6.52.4.6 chFactoryCreateBuffer() ......................... 496
6.52.4.7 chFactoryFindBuffer() ............................ 497
6.52.4.8 chFactoryReleaseBuffer() ....................... 498
6.52.4.9 chFactoryCreateSemaphore() ..................... 498
6.52.4.10 chFactoryFindSemaphore() ....................... 499
6.52.4.11 chFactoryReleaseSemaphore() ................. 499
6.52.4.12 chFactoryCreateMailbox() ...................... 500
6.52.4.13 chFactoryFindMailbox() ......................... 500
6.52.4.14 chFactoryReleaseMailbox() .................... 501
6.52.4.15 chFactoryCreateObjectsFIFO() ................. 501
6.52.4.16 chFactoryFindObjectByPointer() ............... 502
6.52.4.17 chFactoryFindObjectByPointer() ............... 503
6.52.4.18 chFactoryCreatePipe() ......................... 503
6.52.4.19 chFactoryFindPipe() ............................ 504
6.52.4.20 chFactoryReleasePipe() ....................... 504
6.52.4.21 chFactoryDuplicateReference() ............... 506
6.52.4.22 chFactoryGetObject() ......................... 506
6.52.4.23 chFactoryGetBufferSize() ..................... 507
6.52.4.24 chFactoryGetBuffer() .......................... 507
6.52.4.25 chFactoryGetSemaphore() ...................... 508
6.52.4.26 chFactoryGetMailbox() ......................... 508
6.52.4.27 chFactoryGetObjectsFIFO() ................... 509
6.52.4.28 chFactoryGetPipe() ............................ 509

6.52.5 Variable Documentation ............................. 510
6.52.5.1 ch_factory ........................................ 510

7 Data Structure Documentation ................................ 511
7.1 ch_binary_semaphore Struct Reference .................. 511
7.1.1 Detailed Description ................................. 512
7.2 ch_delta_list Struct Reference ....................... 512
7.2.1 Detailed Description ................................ 513
7.2.2 Field Documentation ................................ 513
7.2.2.1 next .............................................. 513
7.2.2.2 prev .............................................. 513
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.2.2.3</td>
<td>delta</td>
<td>513</td>
</tr>
<tr>
<td>7.3</td>
<td>ch_dyn_element Struct Reference</td>
<td>514</td>
</tr>
<tr>
<td>7.3.1</td>
<td>Detailed Description</td>
<td>514</td>
</tr>
<tr>
<td>7.3.2</td>
<td>Field Documentation</td>
<td>514</td>
</tr>
<tr>
<td>7.3.2.1</td>
<td>next</td>
<td>514</td>
</tr>
<tr>
<td>7.3.2.2</td>
<td>refs</td>
<td>514</td>
</tr>
<tr>
<td>7.4</td>
<td>ch_dyn_list Struct Reference</td>
<td>515</td>
</tr>
<tr>
<td>7.4.1</td>
<td>Detailed Description</td>
<td>515</td>
</tr>
<tr>
<td>7.5</td>
<td>ch_dyn_mailbox Struct Reference</td>
<td>515</td>
</tr>
<tr>
<td>7.5.1</td>
<td>Detailed Description</td>
<td>516</td>
</tr>
<tr>
<td>7.5.2</td>
<td>Field Documentation</td>
<td>516</td>
</tr>
<tr>
<td>7.5.2.1</td>
<td>element</td>
<td>517</td>
</tr>
<tr>
<td>7.5.2.2</td>
<td>mbx</td>
<td>517</td>
</tr>
<tr>
<td>7.6</td>
<td>ch_dyn_object Struct Reference</td>
<td>517</td>
</tr>
<tr>
<td>7.6.1</td>
<td>Detailed Description</td>
<td>518</td>
</tr>
<tr>
<td>7.6.2</td>
<td>Field Documentation</td>
<td>518</td>
</tr>
<tr>
<td>7.6.2.1</td>
<td>element</td>
<td>518</td>
</tr>
<tr>
<td>7.7</td>
<td>ch_dyn_objects_fifo Struct Reference</td>
<td>518</td>
</tr>
<tr>
<td>7.7.1</td>
<td>Detailed Description</td>
<td>519</td>
</tr>
<tr>
<td>7.7.2</td>
<td>Field Documentation</td>
<td>519</td>
</tr>
<tr>
<td>7.7.2.1</td>
<td>element</td>
<td>519</td>
</tr>
<tr>
<td>7.7.2.2</td>
<td>fifo</td>
<td>519</td>
</tr>
<tr>
<td>7.8</td>
<td>ch_dyn_pipe Struct Reference</td>
<td>519</td>
</tr>
<tr>
<td>7.8.1</td>
<td>Detailed Description</td>
<td>521</td>
</tr>
<tr>
<td>7.8.2</td>
<td>Field Documentation</td>
<td>521</td>
</tr>
<tr>
<td>7.8.2.1</td>
<td>element</td>
<td>521</td>
</tr>
<tr>
<td>7.8.2.2</td>
<td>pipe</td>
<td>521</td>
</tr>
<tr>
<td>7.9</td>
<td>ch_dyn_semaphore Struct Reference</td>
<td>521</td>
</tr>
<tr>
<td>7.9.1</td>
<td>Detailed Description</td>
<td>522</td>
</tr>
<tr>
<td>7.9.2</td>
<td>Field Documentation</td>
<td>522</td>
</tr>
<tr>
<td>7.9.2.1</td>
<td>element</td>
<td>522</td>
</tr>
<tr>
<td>7.9.2.2</td>
<td>sem</td>
<td>522</td>
</tr>
<tr>
<td>7.10</td>
<td>ch_job_descriptor Struct Reference</td>
<td>522</td>
</tr>
<tr>
<td>7.10.1</td>
<td>Detailed Description</td>
<td>523</td>
</tr>
<tr>
<td>7.10.2</td>
<td>Field Documentation</td>
<td>523</td>
</tr>
<tr>
<td>7.10.2.1</td>
<td>jobfunc</td>
<td>523</td>
</tr>
<tr>
<td>7.10.2.2</td>
<td>jobarg</td>
<td>523</td>
</tr>
<tr>
<td>7.11</td>
<td>ch_jobs_queue Struct Reference</td>
<td>523</td>
</tr>
<tr>
<td>7.11.1</td>
<td>Detailed Description</td>
<td>524</td>
</tr>
<tr>
<td>7.11.2</td>
<td>Field Documentation</td>
<td>524</td>
</tr>
<tr>
<td>7.11.2.1</td>
<td>free</td>
<td>525</td>
</tr>
<tr>
<td>7.11.2.2</td>
<td>mbx</td>
<td>525</td>
</tr>
</tbody>
</table>
7.12 ch_list Struct Reference .................................................. 525
    7.12.1 Detailed Description ................................................ 525
    7.12.2 Field Documentation .............................................. 525
        7.12.2.1 next .................................................. 526
7.13 ch_mutex Struct Reference ............................................... 526
    7.13.1 Detailed Description ............................................... 527
    7.13.2 Field Documentation .............................................. 527
        7.13.2.1 queue ................................................ 527
        7.13.2.2 owner .............................................. 527
        7.13.2.3 next .............................................. 527
        7.13.2.4 cnt ............................................... 527
7.14 ch_objects_cache Struct Reference .................................... 528
    7.14.1 Detailed Description ............................................... 529
    7.14.2 Field Documentation .............................................. 529
        7.14.2.1 hashn .............................................. 529
        7.14.2.2 hashp .............................................. 529
        7.14.2.3 objn ................................................ 530
        7.14.2.4 objsz ............................................... 530
        7.14.2.5 objvp ............................................... 530
        7.14.2.6 lru ................................................ 530
        7.14.2.7 cache_sem ......................................... 530
        7.14.2.8 lru_sem ............................................ 530
        7.14.2.9 readf .............................................. 531
        7.14.2.10 writef ............................................ 531
7.15 ch_objects_factory Struct Reference .................................. 531
    7.15.1 Detailed Description ............................................... 532
    7.15.2 Field Documentation .............................................. 532
        7.15.2.1 mtx ................................................. 532
        7.15.2.2 obj_list .......................................... 532
        7.15.2.3 obj_pool .......................................... 533
        7.15.2.4 buf_list .......................................... 533
        7.15.2.5 sem_list .......................................... 533
        7.15.2.6 sem_pool .......................................... 533
        7.15.2.7 mbx_list .......................................... 533
        7.15.2.8 fifo_list .......................................... 533
        7.15.2.9 pipe_list .......................................... 534
7.16 ch_objects_fifo Struct Reference ..................................... 534
    7.16.1 Detailed Description ............................................... 535
    7.16.2 Field Documentation .............................................. 535
        7.16.2.1 free .............................................. 535
        7.16.2.2 mbx ................................................. 535
7.17 ch_oc_hash_header Struct Reference .................................. 535

ChibiOS/RT
7.17.1 Detailed Description ........................................ 536
7.17.2 Field Documentation ....................................... 536
  7.17.2.1 hash_next ........................................... 537
  7.17.2.2 hash_prev ......................................... 537
7.18 ch_oc_lru_header Struct Reference ......................... 537
  7.18.1 Detailed Description ................................... 538
  7.18.2 Field Documentation ................................... 538
    7.18.2.1 hash_next ....................................... 538
    7.18.2.2 hash_prev ....................................... 538
    7.18.2.3 lru_next ........................................ 538
    7.18.2.4 lru_prev ........................................ 538
7.19 ch_oc_object Struct Reference ................................ 539
  7.19.1 Detailed Description ................................... 540
  7.19.2 Field Documentation ................................... 540
    7.19.2.1 hash_next ....................................... 540
    7.19.2.2 hash_prev ....................................... 540
    7.19.2.3 lru_next ........................................ 540
    7.19.2.4 lru_prev ........................................ 540
    7.19.2.5 obj_group ....................................... 541
    7.19.2.6 obj_key ......................................... 541
    7.19.2.7 obj_sem ......................................... 541
    7.19.2.8 obj_flags ....................................... 541
    7.19.2.9 dptr ............................................. 541
7.20 ch_os_instance Struct Reference ............................. 542
  7.20.1 Detailed Description ................................... 543
  7.20.2 Field Documentation ................................... 543
    7.20.2.1 rlist ............................................ 543
    7.20.2.2 vtlist ........................................... 543
    7.20.2.3 reglist .......................................... 543
    7.20.2.4 core_id ......................................... 543
    7.20.2.5 rfcu ............................................. 544
    7.20.2.6 config ........................................... 544
    7.20.2.7 mainthread ...................................... 544
    7.20.2.8 dbg .............................................. 544
    7.20.2.9 trace_buffer .................................... 544
    7.20.2.10 kernel_stats .................................... 544
7.21 ch_os_instance_config Struct Reference ...................... 545
  7.21.1 Detailed Description ................................... 545
  7.21.2 Field Documentation ................................... 545
    7.21.2.1 name ............................................. 545
    7.21.2.2 mainthread_base ................................ 546
    7.21.2.3 mainthread_end .................................. 546
7.21.2.4 idlethread_base .................................................. 546
7.21.2.5 idlethread_end .................................................. 546
7.22 ch_priority_queue Struct Reference .............................. 546
  7.22.1 Detailed Description ........................................... 547
  7.22.2 Field Documentation ........................................... 547
    7.22.2.1 next ..................................................... 547
    7.22.2.2 prev ..................................................... 547
    7.22.2.3 prio ..................................................... 547
7.23 ch_queue Struct Reference ........................................ 548
  7.23.1 Detailed Description ........................................... 548
  7.23.2 Field Documentation ........................................... 548
    7.23.2.1 next ..................................................... 548
    7.23.2.2 prev ..................................................... 549
7.24 ch_ready_list Struct Reference .................................. 549
  7.24.1 Detailed Description ........................................... 550
  7.24.2 Field Documentation ........................................... 550
    7.24.2.1 pqueue .................................................. 550
    7.24.2.2 current .................................................. 550
7.25 ch_registered_static_object Struct Reference .................. 550
  7.25.1 Detailed Description ........................................... 551
  7.25.2 Field Documentation ........................................... 551
    7.25.2.1 element .................................................. 551
    7.25.2.2 objp ..................................................... 551
7.26 ch_registry Struct Reference .................................... 552
  7.26.1 Detailed Description ........................................... 552
  7.26.2 Field Documentation ........................................... 552
    7.26.2.1 queue .................................................. 552
7.27 ch_rfcu Struct Reference ......................................... 553
  7.27.1 Detailed Description ........................................... 553
  7.27.2 Field Documentation ........................................... 553
    7.27.2.1 mask ..................................................... 553
7.28 ch_semaphore Struct Reference ................................... 554
  7.28.1 Detailed Description ........................................... 555
  7.28.2 Field Documentation ........................................... 555
    7.28.2.1 queue .................................................. 555
    7.28.2.2 cnt ..................................................... 555
7.29 ch_system Struct Reference ...................................... 555
  7.29.1 Detailed Description ........................................... 556
  7.29.2 Field Documentation ........................................... 556
    7.29.2.1 state ................................................... 557
    7.29.2.2 instances .............................................. 557
    7.29.2.3 tmc ..................................................... 557
7.29.2.4 reglist .................................................. 557
7.29.2.5 rfcu .................................................... 557
7.30 ch_system_debug Struct Reference ................................ 558
  7.30.1 Detailed Description ...................................... 558
  7.30.2 Field Documentation ...................................... 558
    7.30.2.1 panic_msg ........................................... 558
    7.30.2.2 isr_cnt ............................................. 559
    7.30.2.3 lock_cnt ........................................... 559
7.31 ch_thread Struct Reference .................................... 559
  7.31.1 Detailed Description ...................................... 561
  7.31.2 Field Documentation ...................................... 561
    7.31.2.1 list ............................................... 561
    7.31.2.2 queue ............................................. 561
    7.31.2.3 pqueue ............................................ 561
    7.31.2.4 hdr ............................................... 562
    7.31.2.5 ctx ............................................... 562
    7.31.2.6 rqueue ............................................ 562
    7.31.2.7 owner ............................................. 562
    7.31.2.8 name .............................................. 562
    7.31.2.9 wabase ............................................ 562
    7.31.2.10 state ............................................. 563
    7.31.2.11 flags ............................................. 563
    7.31.2.12 refs .............................................. 563
    7.31.2.13 ticks ............................................. 563
    7.31.2.14 time .............................................. 563
    7.31.2.15 rdymsg .......................................... 564
    7.31.2.16 exitcode ........................................ 564
    7.31.2.17 wtojb ........................................... 564
    7.31.2.18 wttrp ............................................ 564
    7.31.2.19 sentmsg ......................................... 565
    7.31.2.20 wtsemp ........................................... 565
    7.31.2.21 wtmtxp .......................................... 565
    7.31.2.22 ewmask .......................................... 565
    7.31.2.23 u ................................................ 566
    7.31.2.24 waiting ......................................... 566
    7.31.2.25 msgqueue ........................................ 566
    7.31.2.26 epending ........................................ 566
    7.31.2.27 mtxlist .......................................... 566
    7.31.2.28 realprio ......................................... 567
    7.31.2.29 mpool ............................................ 567
    7.31.2.30 stats ............................................. 567
7.32 ch_threads_queue Struct Reference ............................... 567
7.32.1 Detailed Description ........................................ 568
7.32.2 Field Documentation ....................................... 568
  7.32.2.1 queue .................................................. 568
7.33 ch_virtual_timer Struct Reference .......................... 568
  7.33.1 Detailed Description .................................... 569
  7.33.2 Field Documentation ..................................... 569
    7.33.2.1 dlist ............................................... 569
    7.33.2.2 func ................................................. 569
    7.33.2.3 par ................................................ 569
    7.33.2.4 reload .............................................. 569
7.34 ch_virtual_timers_list Struct Reference .................. 570
  7.34.1 Detailed Description .................................... 570
  7.34.2 Field Documentation ..................................... 571
    7.34.2.1 dlist ............................................... 571
    7.34.2.2 systime ............................................ 571
    7.34.2.3 lasttime ............................................ 571
    7.34.2.4 laststamp .......................................... 571
7.35 chdebug_t Struct Reference .................................. 572
  7.35.1 Detailed Description .................................... 574
  7.35.2 Field Documentation ..................................... 574
    7.35.2.1 identifier .......................................... 574
    7.35.2.2 zero ............................................... 574
    7.35.2.3 size ............................................... 574
    7.35.2.4 version ............................................ 575
    7.35.2.5 ptrsize ............................................ 575
    7.35.2.6 timesize ........................................... 575
    7.35.2.7 threadsize ......................................... 575
    7.35.2.8 off_prio ........................................... 575
    7.35.2.9 off_ctx ............................................ 575
    7.35.2.10 off_newer .......................................... 576
    7.35.2.11 off_older .......................................... 576
    7.35.2.12 off_name .......................................... 576
    7.35.2.13 off_stklimit ....................................... 576
    7.35.2.14 off_state ......................................... 576
    7.35.2.15 off_flags ......................................... 576
    7.35.2.16 off.refs .......................................... 577
    7.35.2.17 off_preempt ....................................... 577
    7.35.2.18 off_time ......................................... 577
    7.35.2.19 intctxsize ........................................ 577
    7.35.2.20 intervalsize ....................................... 577
    7.35.2.21 instancesnum ....................................... 577
    7.35.2.22 off.sys_state ..................................... 578
7.46.2.9 cmtx ................................................. 600
7.46.2.10 wmtx ............................................. 600
7.46.2.11 rmtx ............................................. 600

7.47 pool_header Struct Reference ............................................. 600

7.47.1 Detailed Description ............................................. 601
7.47.2 Field Documentation ............................................. 601

7.47.2.1 next ............................................. 601

7.48 thread_descriptor_t Struct Reference ............................................. 601

7.48.1 Detailed Description ............................................. 602
7.48.2 Field Documentation ............................................. 602

7.48.2.1 name ............................................. 602
7.48.2.2 wbase ............................................. 602
7.48.2.3 wend ............................................. 602
7.48.2.4 prio ............................................. 603
7.48.2.5 funcp ............................................. 603
7.48.2.6 arg ............................................. 603
7.48.2.7 instance ............................................. 603

7.49 time_measurement_t Struct Reference ............................................. 603

7.49.1 Detailed Description ............................................. 604
7.49.2 Field Documentation ............................................. 604

7.49.2.1 best ............................................. 604
7.49.2.2 worst ............................................. 604
7.49.2.3 last ............................................. 605
7.49.2.4 n ............................................. 605
7.49.2.5 cumulative ............................................. 605

7.50 tm_calibration_t Struct Reference ............................................. 605

7.50.1 Detailed Description ............................................. 606
7.50.2 Field Documentation ............................................. 606

7.50.2.1 offset ............................................. 606

7.51 trace_buffer_t Struct Reference ............................................. 606

7.51.1 Detailed Description ............................................. 608
7.51.2 Field Documentation ............................................. 608

7.51.2.1 suspended ............................................. 608
7.51.2.2 size ............................................. 608
7.51.2.3 ptr ............................................. 608
7.51.2.4 buffer ............................................. 608

7.52 trace_event_t Struct Reference ............................................. 609

7.52.1 Detailed Description ............................................. 610
7.52.2 Field Documentation ............................................. 611

7.52.2.1 type ............................................. 611
7.52.2.2 state ............................................. 611
7.52.2.3 rtstamp ............................................. 611
7.52.2.4 time ................................................................. 611
7.52.2.5 ntp ................................................................. 611
7.52.2.6 wtobjp .............................................................. 612
7.52.2.7 sw ................................................................. 612
7.52.2.8 tp ................................................................. 612
7.52.2.9 msg ................................................................. 612
7.52.2.10 rdy ............................................................... 612
7.52.2.11 name ............................................................. 612
7.52.2.12 isr ............................................................... 613
7.52.2.13 reason .......................................................... 613
7.52.2.14 halt .............................................................. 613
7.52.2.15 up1 .............................................................. 613
7.52.2.16 up2 .............................................................. 613
7.52.2.17 user .............................................................. 613

8 File Documentation 615

8.1 ch.h File Reference .................................................. 615
  8.1.1 Detailed Description ........................................... 616
8.2 chalign.h File Reference .......................................... 616
  8.2.1 Detailed Description ........................................... 616
8.3 chbsem.h File Reference .......................................... 617
  8.3.1 Detailed Description ........................................... 618
8.4 chchecks.h File Reference ........................................ 618
  8.4.1 Detailed Description ........................................... 618
8.5 chcond.c File Reference ........................................... 618
  8.5.1 Detailed Description ........................................... 619
8.6 chcond.h File Reference .......................................... 619
  8.6.1 Detailed Description ........................................... 620
8.7 chconf.h File Reference .......................................... 620
  8.7.1 Detailed Description ........................................... 623
8.8 chcustomer.h File Reference ..................................... 623
  8.8.1 Detailed Description ........................................... 624
8.9 chdebug.c File Reference ........................................ 624
  8.9.1 Detailed Description ........................................... 625
8.10 chdebug.h File Reference ....................................... 625
  8.10.1 Detailed Description .......................................... 626
8.11 chdelegates.c File Reference .................................. 626
  8.11.1 Detailed Description .......................................... 626
8.12 chdelegates.h File Reference .................................. 627
  8.12.1 Detailed Description .......................................... 628
8.13 chdynamic.c File Reference .................................... 628
  8.13.1 Detailed Description .......................................... 628
<table>
<thead>
<tr>
<th>Section</th>
<th>File Reference</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>8.14</td>
<td>chdynamic.h</td>
<td>628</td>
</tr>
<tr>
<td></td>
<td>8.14.1 Detailed Description</td>
<td>628</td>
</tr>
<tr>
<td>8.15</td>
<td>chearly.h</td>
<td>629</td>
</tr>
<tr>
<td></td>
<td>8.15.1 Detailed Description</td>
<td>630</td>
</tr>
<tr>
<td>8.16</td>
<td>chevents.c</td>
<td>630</td>
</tr>
<tr>
<td></td>
<td>8.16.1 Detailed Description</td>
<td>631</td>
</tr>
<tr>
<td>8.17</td>
<td>chevents.h</td>
<td>631</td>
</tr>
<tr>
<td></td>
<td>8.17.1 Detailed Description</td>
<td>632</td>
</tr>
<tr>
<td>8.18</td>
<td>chfactory.c</td>
<td>633</td>
</tr>
<tr>
<td></td>
<td>8.18.1 Detailed Description</td>
<td>634</td>
</tr>
<tr>
<td>8.19</td>
<td>chfactory.h</td>
<td>634</td>
</tr>
<tr>
<td></td>
<td>8.19.1 Detailed Description</td>
<td>636</td>
</tr>
<tr>
<td>8.20</td>
<td>chinstances.c</td>
<td>637</td>
</tr>
<tr>
<td></td>
<td>8.20.1 Detailed Description</td>
<td>637</td>
</tr>
<tr>
<td>8.21</td>
<td>chinstances.h</td>
<td>637</td>
</tr>
<tr>
<td></td>
<td>8.21.1 Detailed Description</td>
<td>637</td>
</tr>
<tr>
<td>8.22</td>
<td>chjobs.h</td>
<td>637</td>
</tr>
<tr>
<td></td>
<td>8.22.1 Detailed Description</td>
<td>639</td>
</tr>
<tr>
<td>8.23</td>
<td>chlib.h</td>
<td>639</td>
</tr>
<tr>
<td></td>
<td>8.23.1 Detailed Description</td>
<td>640</td>
</tr>
<tr>
<td>8.24</td>
<td>chlicense.h</td>
<td>640</td>
</tr>
<tr>
<td></td>
<td>8.24.1 Detailed Description</td>
<td>641</td>
</tr>
<tr>
<td>8.25</td>
<td>chlists.h</td>
<td>641</td>
</tr>
<tr>
<td></td>
<td>8.25.1 Detailed Description</td>
<td>643</td>
</tr>
<tr>
<td>8.26</td>
<td>chmboxes.c</td>
<td>643</td>
</tr>
<tr>
<td></td>
<td>8.26.1 Detailed Description</td>
<td>643</td>
</tr>
<tr>
<td>8.27</td>
<td>chmboxes.h</td>
<td>643</td>
</tr>
<tr>
<td></td>
<td>8.27.1 Detailed Description</td>
<td>645</td>
</tr>
<tr>
<td>8.28</td>
<td>chmemcore.c</td>
<td>645</td>
</tr>
<tr>
<td></td>
<td>8.28.1 Detailed Description</td>
<td>645</td>
</tr>
<tr>
<td>8.29</td>
<td>chmemcore.h</td>
<td>645</td>
</tr>
<tr>
<td></td>
<td>8.29.1 Detailed Description</td>
<td>646</td>
</tr>
<tr>
<td>8.30</td>
<td>chmemheaps.c</td>
<td>646</td>
</tr>
<tr>
<td></td>
<td>8.30.1 Detailed Description</td>
<td>647</td>
</tr>
<tr>
<td>8.31</td>
<td>chmemheaps.h</td>
<td>647</td>
</tr>
<tr>
<td></td>
<td>8.31.1 Detailed Description</td>
<td>648</td>
</tr>
<tr>
<td>8.32</td>
<td>chmempools.c</td>
<td>648</td>
</tr>
<tr>
<td></td>
<td>8.32.1 Detailed Description</td>
<td>649</td>
</tr>
<tr>
<td>8.33</td>
<td>chmempools.h</td>
<td>649</td>
</tr>
<tr>
<td></td>
<td>8.33.1 Detailed Description</td>
<td>650</td>
</tr>
<tr>
<td>8.34</td>
<td>chmsg.c</td>
<td>650</td>
</tr>
<tr>
<td></td>
<td>8.34.1 Detailed Description</td>
<td>651</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

Module Index

2.1 Modules

Here is a list of all modules:

- Release and Licensing .................................................. 15
- Release Information ....................................................... 16
- Customer Information ..................................................... 18
- License Settings ............................................................ 20
- RT Kernel ........................................................................ 22
- Version Numbers and Identification .................................... 23
- Configuration .................................................................. 25
- Options ........................................................................... 26
- Checks ............................................................................. 48
- Restrictions ...................................................................... 49
- System ............................................................................ 50
- Port Interface .................................................................... 51
- OS Types and Structures ................................................... 52
- OS Instances .................................................................... 63
- Runtime Faults Collection Unit .......................................... 66
- Lists and Queues .............................................................. 69
- Scheduler ......................................................................... 84
- Base Kernel Services .......................................................... 105
- System Management ............................................................ 106
- Memory Alignment ............................................................. 131
- Time and Intervals .............................................................. 134
- Virtual Timers .................................................................. 148
- Threads ............................................................................ 177
- Time Measurement ............................................................. 220
- Synchronization ................................................................ 224
- Counting Semaphores ......................................................... 225
- Mutexes ............................................................................ 243
- Condition Variables ............................................................ 258
- Event Flags ....................................................................... 270
- Synchronous Messages ....................................................... 297
- Dynamic Threads ................................................................. 308
- Registry ............................................................................ 311
- Debug ............................................................................... 321
- Checks and Assertions ........................................................ 322
<table>
<thead>
<tr>
<th>Module Index</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tracing</td>
<td>332</td>
</tr>
<tr>
<td>Statistics</td>
<td>342</td>
</tr>
<tr>
<td>OS Library</td>
<td>346</td>
</tr>
<tr>
<td>Version Numbers and Identification</td>
<td>347</td>
</tr>
<tr>
<td>Synchronization</td>
<td>350</td>
</tr>
<tr>
<td>Binary Semaphores</td>
<td>351</td>
</tr>
<tr>
<td>Mailboxes</td>
<td>362</td>
</tr>
<tr>
<td>Pipes</td>
<td>380</td>
</tr>
<tr>
<td>Delegate Threads</td>
<td>388</td>
</tr>
<tr>
<td>Jobs Queues</td>
<td>400</td>
</tr>
<tr>
<td>Memory Management</td>
<td>415</td>
</tr>
<tr>
<td>Core Memory Manager</td>
<td>416</td>
</tr>
<tr>
<td>Memory Heaps</td>
<td>427</td>
</tr>
<tr>
<td>Memory Pools</td>
<td>434</td>
</tr>
<tr>
<td>Complex Services</td>
<td>459</td>
</tr>
<tr>
<td>Objects FIFOs</td>
<td>460</td>
</tr>
<tr>
<td>Objects Caches</td>
<td>477</td>
</tr>
<tr>
<td>Dynamic Objects Factory</td>
<td>487</td>
</tr>
</tbody>
</table>
Chapter 3

Hierarchical Index

3.1 Class Hierarchy

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

ch_delta_list .................................................. 512
ch_dyn_element .................................................. 514
ch_dyn_list ..................................................... 515
ch_dyn_mailbox .................................................. 515
ch_dyn_object .................................................... 517
ch_dyn_objects_filo ............................................. 518
ch_dyn_pipe ..................................................... 519
ch_dyn_semaphore ............................................... 521
ch_job_descriptor ............................................... 522
ch_jobs_queue ................................................... 523
ch_list .......................................................... 525
ch_mutex ......................................................... 526
ch_objects_cache ............................................... 528
ch_objects_factory ............................................. 531
ch_objects_fio ................................................... 534
ch_oc_hash_header ............................................. 535
ch_oc_ifr_header ................................................ 537
ch_oc_object .................................................... 539
ch_os_instance .................................................. 542
ch_os_instance_config ......................................... 545
ch_priority_queue .............................................. 546
ch_queue ........................................................ 548
ch_ready_list .................................................... 549
ch_registered_static_object ................................. 550
ch_registry ...................................................... 552
ch_rfcu .......................................................... 553
ch_semaphore .................................................... 554
    ch_binary_semaphore ........................................ 511
ch_system ....................................................... 555
ch_system_debug ............................................... 558
ch_thread ....................................................... 559
ch_threads_queue ................................................ 567
ch_virtual_timer ............................................... 568
ch_virtual_timers_list ........................................ 570
chdebug_t ....................................................... 572
<table>
<thead>
<tr>
<th>Symbol</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>condition_variable</td>
<td>579</td>
</tr>
<tr>
<td>event_listener</td>
<td>580</td>
</tr>
<tr>
<td>event_source</td>
<td>582</td>
</tr>
<tr>
<td>guarded_memory_pool_t</td>
<td>584</td>
</tr>
<tr>
<td>heap_header</td>
<td>585</td>
</tr>
<tr>
<td>kernel_stats_t</td>
<td>587</td>
</tr>
<tr>
<td>mailbox_t</td>
<td>589</td>
</tr>
<tr>
<td>memcore_t</td>
<td>592</td>
</tr>
<tr>
<td>memory_heap</td>
<td>593</td>
</tr>
<tr>
<td>memory_pool_t</td>
<td>595</td>
</tr>
<tr>
<td>pipe_t</td>
<td>596</td>
</tr>
<tr>
<td>pool_header</td>
<td>600</td>
</tr>
<tr>
<td>thread_descriptor_t</td>
<td>601</td>
</tr>
<tr>
<td>time_measurement_t</td>
<td>603</td>
</tr>
<tr>
<td>tm_calibration_t</td>
<td>605</td>
</tr>
<tr>
<td>trace_buffer_t</td>
<td>606</td>
</tr>
<tr>
<td>trace_event_t</td>
<td>609</td>
</tr>
</tbody>
</table>
4.1 Data Structures

Here are the data structures with brief descriptions:

- **ch_binary_semaphore**: Binary semaphore type .................................................. 511
- **ch_delta_list**: Delta list element and header structure ........................................ 512
- **ch_dyn_element**: Type of a dynamic object list element ...................................... 514
- **ch_dyn_list**: Type of a dynamic object list .......................................................... 515
- **ch_dyn_mailbox**: Type of a dynamic buffer object ............................................... 515
- **ch_dyn_object**: Type of a dynamic buffer object .................................................. 517
- **ch_dyn_objects_fifo**: Type of a dynamic buffer object .......................................... 518
- **ch_dyn_pipe**: Type of a dynamic pipe object ....................................................... 519
- **ch_dyn_semaphore**: Type of a dynamic semaphore .............................................. 521
- **ch_job_descriptor**: Type of a job descriptor ....................................................... 522
- **ch_jobs_queue**: Type of a jobs queue ................................................................. 523
- **ch_list**: Structure representing a generic single link list header and element .......... 525
- **ch_mutex**: Mutex structure ................................................................................... 526
- **ch_objects_cache**: Structure representing a cache object .................................... 528
- **ch_objects_factory**: Type of the factory main object ........................................... 531
- **ch_objects_fifo**: Type of an objects FIFO ............................................................ 534
- **ch_oc_hash_header**: Structure representing an hash table element ....................... 535
- **ch_oc_lru_header**: Structure representing an hash table element .......................... 537
<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</td>
</tr>
<tr>
<td><strong>ch_os_instance</strong></td>
</tr>
<tr>
<td>System instance data structure</td>
</tr>
<tr>
<td><strong>ch_os_instance_config</strong></td>
</tr>
<tr>
<td>Type of an system instance configuration</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</td>
</tr>
<tr>
<td><strong>ch_queue</strong></td>
</tr>
<tr>
<td>Structure representing a generic bidirectional linked list header and element</td>
</tr>
<tr>
<td><strong>ch_ready_list</strong></td>
</tr>
<tr>
<td>Type of a ready list header</td>
</tr>
<tr>
<td><strong>ch_registered_static_object</strong></td>
</tr>
<tr>
<td>Type of a registered object</td>
</tr>
<tr>
<td><strong>ch_registry</strong></td>
</tr>
<tr>
<td>Type of a registry structure</td>
</tr>
<tr>
<td><strong>ch_rfu</strong></td>
</tr>
<tr>
<td>Type of an RFCU structure</td>
</tr>
<tr>
<td><strong>ch_semaphore</strong></td>
</tr>
<tr>
<td>Semaphore structure</td>
</tr>
<tr>
<td><strong>ch_system</strong></td>
</tr>
<tr>
<td>Type of system data structure</td>
</tr>
<tr>
<td><strong>ch_system_debug</strong></td>
</tr>
<tr>
<td>System debug data structure</td>
</tr>
<tr>
<td><strong>ch_thread</strong></td>
</tr>
<tr>
<td>Structure representing a thread</td>
</tr>
<tr>
<td><strong>ch_threads_queue</strong></td>
</tr>
<tr>
<td>Type of a threads queue</td>
</tr>
<tr>
<td><strong>ch_virtual_timer</strong></td>
</tr>
<tr>
<td>Structure representing a Virtual Timer</td>
</tr>
<tr>
<td><strong>ch_virtual_timers_list</strong></td>
</tr>
<tr>
<td>Type of virtual timers list header</td>
</tr>
<tr>
<td><strong>chdebug_t</strong></td>
</tr>
<tr>
<td>ChibiOS/RT memory signature record</td>
</tr>
<tr>
<td><strong>condition_variable</strong></td>
</tr>
<tr>
<td>Condition_variable_t structure</td>
</tr>
<tr>
<td><strong>event_listener</strong></td>
</tr>
<tr>
<td>Event Listener structure</td>
</tr>
<tr>
<td><strong>event_source</strong></td>
</tr>
<tr>
<td>Event Source structure</td>
</tr>
<tr>
<td><strong>guarded_memory_pool_t</strong></td>
</tr>
<tr>
<td>Guarded memory pool descriptor</td>
</tr>
<tr>
<td><strong>heap_header</strong></td>
</tr>
<tr>
<td>Memory heap block header</td>
</tr>
<tr>
<td><strong>kernel_stats_t</strong></td>
</tr>
<tr>
<td>Type of a kernel statistics structure</td>
</tr>
<tr>
<td><strong>mailbox_t</strong></td>
</tr>
<tr>
<td>Structure representing a mailbox object</td>
</tr>
<tr>
<td><strong>memcore_t</strong></td>
</tr>
<tr>
<td>Type of memory core object</td>
</tr>
<tr>
<td><strong>memory_heap</strong></td>
</tr>
<tr>
<td>Structure describing a memory heap</td>
</tr>
<tr>
<td><strong>memory_pool_t</strong></td>
</tr>
<tr>
<td>Memory pool descriptor</td>
</tr>
<tr>
<td><strong>pipe_t</strong></td>
</tr>
<tr>
<td>Structure representing a pipe object</td>
</tr>
<tr>
<td><strong>pool_header</strong></td>
</tr>
<tr>
<td>Memory pool free object header</td>
</tr>
</tbody>
</table>
4.1 Data Structures

thread_descriptor_t
Type of a thread descriptor ........................................ 601

time_measurement_t
Type of a Time Measurement object .................................. 603

tm_calibration_t
Type of a time measurement calibration data ....................... 605

trace_buffer_t
Trace buffer header ..................................................... 606

trace_event_t
Trace buffer record ..................................................... 609
Chapter 5

File Index

5.1 File List

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

- ch.h ChibiOS/RT main include file ................................................................. 615
- chalign.h Memory alignment macros and structures ........................................ 616
- chbsem.h Binary semaphores structures and macros ....................................... 617
- chchecks.h Configuration file checks header ............................................... 618
- chcond.c Condition Variables code ............................................................... 618
- chcond.h Condition Variables macros and structures ...................................... 619
- chconf.h Configuration file template ............................................................ 620
- chcustomer.h Customer-related info ............................................................ 623
- chdebug.c Debug support code ....................................................................... 624
- chdebug.h Debug support macros and structures ............................................ 625
- chdelegates.c Delegate threads code ............................................................. 626
- chdelegates.h Delegate threads macros and structures ................................... 627
- chdynamic.c Dynamic threads code ............................................................... 628
- chdynamic.h Dynamic threads macros and structures .................................... 628
- chearly.h Early forward types declarations header ....................................... 629
- chevents.c Events code ................................................................................. 630
- chevents.h Events macros and structures ....................................................... 631
- chfactory.c ChibiOS objects factory and registry code ................................... 633
<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>chrfcu.c</td>
<td>Runtime Faults Collection Unit code</td>
<td>662</td>
</tr>
<tr>
<td>chrfcu.h</td>
<td>Runtime Faults Collection Unit macros and structures</td>
<td>663</td>
</tr>
<tr>
<td>chschd.c</td>
<td>Scheduler code</td>
<td>664</td>
</tr>
<tr>
<td>chschd.h</td>
<td>Scheduler macros and structures</td>
<td>664</td>
</tr>
<tr>
<td>chsem.c</td>
<td>Semaphores code</td>
<td>667</td>
</tr>
<tr>
<td>chsem.h</td>
<td>Semaphores macros and structures</td>
<td>668</td>
</tr>
<tr>
<td>chstats.c</td>
<td>Statistics module code</td>
<td>669</td>
</tr>
<tr>
<td>chstats.h</td>
<td>Statistics module macros and structures</td>
<td>670</td>
</tr>
<tr>
<td>chsys.c</td>
<td>System related code</td>
<td>670</td>
</tr>
<tr>
<td>chsys.h</td>
<td>System related macros and structures</td>
<td>671</td>
</tr>
<tr>
<td>chthreads.c</td>
<td>Threads code</td>
<td>673</td>
</tr>
<tr>
<td>chthreads.h</td>
<td>Threads module macros and structures</td>
<td>675</td>
</tr>
<tr>
<td>chtime.h</td>
<td>Time and intervals macros and structures</td>
<td>678</td>
</tr>
<tr>
<td>chtm.c</td>
<td>Time Measurement module code</td>
<td>679</td>
</tr>
<tr>
<td>chtm.h</td>
<td>Time Measurement module macros and structures</td>
<td>680</td>
</tr>
<tr>
<td>chtrace.c</td>
<td>Tracer code</td>
<td>680</td>
</tr>
<tr>
<td>chtrace.h</td>
<td>Tracer macros and structures</td>
<td>681</td>
</tr>
<tr>
<td>chversion.h</td>
<td>Version Module macros and structures</td>
<td>683</td>
</tr>
<tr>
<td>chvt.c</td>
<td>Time and Virtual Timers module code</td>
<td>683</td>
</tr>
<tr>
<td>chvt.h</td>
<td>Time and Virtual Timers module macros and structures</td>
<td>684</td>
</tr>
</tbody>
</table>
Chapter 6

Module Documentation

6.1 Release and Licensing

6.1.1 Detailed Description

Modules

• Release Information
• Customer Information
• License Settings
6.2 Release Information

6.2.1 Detailed Description

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

Macros

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

ChibiOS version identification

- `#define CH_VERSION "2021.11.0"`
  ChibiOS version string.
- `#define CH_VERSION_YEAR 21`
  ChibiOS version release year.
- `#define CH_VERSION_MONTH 11`
  ChibiOS version release month.
- `#define CH_VERSION_PATCH 0`
  ChibiOS version patch number.
- `#define CH_VERSION_NICKNAME "Vettica"`
  ChibiOS version nickname.

6.2.2 Macro Definition Documentation

6.2.2.1 __CHIBIOS__

```
#define __CHIBIOS__
```

ChibiOS product identification macro.

6.2.2.2 CH_VERSION_STABLE

```
#define CH_VERSION_STABLE 1
```

Stable release flag.
6.2 Release Information

6.2.2.3 CH_VERSION

#define CH_VERSION "2021.11.0"

ChibiOS version string.

6.2.2.4 CH_VERSION_YEAR

#define CH_VERSION_YEAR 21

ChibiOS version release year.

6.2.2.5 CH_VERSION_MONTH

#define CH_VERSION_MONTH 11

ChibiOS version release month.

6.2.2.6 CH_VERSION_PATCH

#define CH_VERSION_PATCH 0

ChibiOS version patch number.

6.2.2.7 CH_VERSION_NICKNAME

#define CH_VERSION_NICKNAME "Vettica"

ChibiOS version nickname.

6.2.2.8 CH_VERSION_DATE

#define CH_VERSION_DATE {{{(CH_VERSION_YEAR + 2000) * 100} + CH_VERSION_MONTH}

Current version date in numeric form (yyyymm).
6.3 Customer Information

6.3.1 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_CUSTOMER_LICENSE_EOS_DATE 209912
  End-Of-Support date (yyyyyy).
- #define CH_CUSTOMER_LICENSE_VERSION_YEAR 99
  Licensed branch year.
- #define CH_CUSTOMER_LICENSE_VERSION_MONTH 12
  Licensed branch month.
- #define CH_LICENSE CH_LICENSE_GPL
  Current license.
- #define CH_CUSTOMER_LICENSE_VERSION_DATE
  Licensed version date in numeric form (yyyyyy).

Licensed Products

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

6.3.2 Macro Definition Documentation
6.3.2.1  CH_CUSTOMER_ID_STRING

#define CH_CUSTOMER_ID_STRING "Santa, North Pole"
Customer readable identifier.

6.3.2.2  CH_CUSTOMER_ID_CODE

#define CH_CUSTOMER_ID_CODE "xxxx-yyyy"
Customer code.

6.3.2.3  CH_CUSTOMER_LICENSE_EOS_DATE

#define CH_CUSTOMER_LICENSE_EOS_DATE 209912
End-Of-Support date (yyyymm).

6.3.2.4  CH_CUSTOMER_LICENSE_VERSION_YEAR

#define CH_CUSTOMER_LICENSE_VERSION_YEAR 99
Licensed branch year.

6.3.2.5  CH_CUSTOMER_LICENSE_VERSION_MONTH

#define CH_CUSTOMER_LICENSE_VERSION_MONTH 12
Licensed branch month.

6.3.2.6  CH_LICENSE

#define CH_LICENSE CH_LICENSE_GPL
Current license.

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

6.3.2.7  CH_CUSTOMER_LICENSE_VERSION_DATE

#define CH_CUSTOMER_LICENSE_VERSION_DATE
Value:
{{(CH_CUSTOMER_LICENSE_VERSION_YEAR + 2000) * 100) +
  CH_CUSTOMER_LICENSE_VERSION_MONTH}
Licensed version date in numeric form (yyyymm).
6.4 License Settings

6.4.1 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`

6.4.2 Macro Definition Documentation
6.4.2.1  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.

6.4.2.2  CH_LICENSE_ID_STRING

#define CH_LICENSE_ID_STRING "N/A"

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

6.4.2.3  CH_LICENSE_ID_CODE

#define CH_LICENSE_ID_CODE "N/A"

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

6.4.2.4  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.

6.4.2.5  CH_LICENSE_FEATURES

#define CH_LICENSE_FEATURES CH_FEATURES_FULL

Code functionality restrictions.

6.4.2.6  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.
6.5 RT Kernel

6.5.1 Detailed Description

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

Modules

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

6.6.1 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 "7.0.1"
  
  Kernel version string.

- \#define CH\_KERNEL\_MAJOR 7
  
  Kernel version major number.

- \#define CH\_KERNEL\_MINOR 0
  
  Kernel version minor number.

- \#define CH\_KERNEL\_PATCH 1
  
  Kernel version patch number.

Constants for configuration options

- \#define FALSE 0
  
  Generic "false" preprocessor boolean constant.

- \#define TRUE 1
  
  Generic "true" preprocessor boolean constant.

6.6.2 Macro Definition Documentation

6.6.2.1 \_CHIBIOS\_RT\_

\#define \_CHIBIOS\_RT\_

ChibiOS/RT identification macro.
6.6.2.2 CH_KERNEL_STABLE

#define CH_KERNEL_STABLE 1
Stable release flag.

6.6.2.3 CH_KERNEL_VERSION

#define CH_KERNEL_VERSION "7.0.1"
Kernel version string.

6.6.2.4 CH_KERNEL_MAJOR

#define CH_KERNEL_MAJOR 7
Kernel version major number.

6.6.2.5 CH_KERNEL_MINOR

#define CH_KERNEL_MINOR 0
Kernel version minor number.

6.6.2.6 CH_KERNEL_PATCH

#define CH_KERNEL_PATCH 1
Kernel version patch number.

6.6.2.7 FALSE

#define FALSE 0
Generic 'false' preprocessor boolean constant.

Note

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

6.6.2.8 TRUE

#define TRUE 1
Generic 'true' preprocessor boolean constant.

Note

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

6.7.1 Detailed Description

Modules

• Options
• Checks
• Restrictions
6.8 Options

6.8.1 Detailed Description

Kernel related settings and hooks.

System settings

- \#define CH_CFG_SMP_MODE FALSE
  Handling of instances.

System timers settings

- \#define CH_CFG_ST_RESOLUTION 32
  System time counter resolution.
- \#define CH_CFG_ST_FREQUENCY 10000
  System tick frequency.
- \#define CH_CFG_INTERVALS_SIZE 32
  Time intervals data size.
- \#define CH_CFG_TIME_TYPES_SIZE 32
  Time types data size.
- \#define CH_CFG_ST_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.
Subsystems options

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

OSLIB options

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

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

Debug options

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

Kernel hooks

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

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

6.8.2 Macro Definition Documentation

6.8.2.1 CH_CFG_SMP_MODE

#define CH_CFG_SMP_MODE FALSE

Handling of instances.

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

#define CH_CFG_ST_RESOLUTION 32

System time counter resolution.

Note
   Allowed values are 16, 32 or 64 bits.

6.8.2.3 CH_CFG_ST_FREQUENCY

#define CH_CFG_ST_FREQUENCY 10000

System tick frequency.

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

6.8.2.4 CH_CFG_INTERVALS_SIZE

#define CH_CFG_INTERVALS_SIZE 32

Time intervals data size.

Note
   Allowed values are 16, 32 or 64 bits.

6.8.2.5 CH_CFG_TIME_TYPES_SIZE

#define CH_CFG_TIME_TYPES_SIZE 32

Time types data size.

Note
   Allowed values are 16 or 32 bits.
6.8.2.6 CH_CFG_ST_TIMEDELTA

#define CH_CFG_ST_TIMEDELTA 2

Time delta constant for the tick-less mode.

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

6.8.2.7 CH_CFG_TIME_QUANTUM

#define CH_CFG_TIME_QUANTUM 0

Round robin interval.

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

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

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

6.8.2.8 CH_CFG_NO_IDLE_THREAD

#define CH_CFG_NO_IDLE_THREAD FALSE

Idle thread automatic spawn suppression.

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

6.8.2.9 CH_CFG_OPTIMIZE_SPEED

#define CH_CFG_OPTIMIZE_SPEED TRUE

OS optimization.

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

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

#define CH_CFG_USE_TM TRUE

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

Note
  The default is TRUE.

6.8.2.11  CH_CFG_USE_TIMESTAMP

#define CH_CFG_USE_TIMESTAMP TRUE

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

Note
  The default is TRUE.

6.8.2.12  CH_CFG_USE_REGISTRY

#define CH_CFG_USE_REGISTRY TRUE

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

Note
  The default is TRUE.

6.8.2.13  CH_CFG_USE_WAITEXIT

#define CH_CFG_USE_WAITEXIT TRUE

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

Note
  The default is TRUE.
6.8 Options

6.8.2.14 CH_CFG_USE_SEMAPHORES

#define CH_CFG_USE_SEMAPHORES TRUE

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

Note
The default is TRUE.

6.8.2.15 CH_CFG_USE_SEMAPHORES_PRIORITY

#define CH_CFG_USE_SEMAPHORES_PRIORITY FALSE

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

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

6.8.2.16 CH_CFG_USE_MUTEXES

#define CH_CFG_USE_MUTEXES TRUE

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

Note
The default is TRUE.

6.8.2.17 CH_CFG_USE_MUTEXES_RECURSIVE

#define CH_CFG_USE_MUTEXES_RECURSIVE FALSE

Enables recursive behavior on mutexes.

Note
Recursive mutexes are heavier and have an increased memory footprint.
The default is FALSE.
Requires CH_CFG_USE_MUTEXES.
6.8.2.18 CH_CFG_USE_CONDVARS

#define CH_CFG_USE_CONDVARS TRUE

Conditional Variables APIs.

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

Note

The default is TRUE.
Requires CH_CFG_USE_MUTEXES.

6.8.2.19 CH_CFG_USE_CONDVARS_TIMEOUT

#define CH_CFG_USE_CONDVARS_TIMEOUT TRUE

Conditional Variables APIs with timeout.

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

Note

The default is TRUE.
Requires CH_CFG_USE_CONDVARS.

6.8.2.20 CH_CFG_USE_EVENTS

#define CH_CFG_USE_EVENTS TRUE

Events Flags APIs.

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

Note

The default is TRUE.
6.8.2.21 CH_CFG_USE_EVENTS_TIMEOUT

#define CH_CFG_USE_EVENTS_TIMEOUT TRUE

Events Flags APIs with timeout.

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

Note
The default is TRUE.
Requires CH_CFG_USE_EVENTS.

6.8.2.22 CH_CFG_USE_MESSAGES

#define CH_CFG_USE_MESSAGES TRUE

Synchronous Messages APIs.

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

Note
The default is TRUE.

6.8.2.23 CH_CFG_USE_MESSAGES_PRIORITY

#define CH_CFG_USE_MESSAGES_PRIORITY FALSE

Synchronous Messages queuing mode.

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

Note
The default is FALSE. Enable this if you have special requirements.
Requires CH_CFG_USE_MESSAGES.
**6.8.2.24 CH_CFG_USE_DYNAMIC**

#define CH_CFG_USE_DYNAMIC TRUE

Dynamic Threads APIs.

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

**Note**

The default is TRUE.

Requires CH_CFG_USE_WAITEXIT.

Requires CH_CFG_USE_HEAP and/or CH_CFG_USE_MEMPOOLS.

---

**6.8.2.25 CH_CFG_USE_MAILBOXES**

#define CH_CFG_USE_MAILBOXES TRUE

Mailboxes APIs.

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

**Note**

The default is TRUE.

Requires CH_CFG_USE_SEMAPHORES.

---

**6.8.2.26 CH_CFG_USE_MEMCORE**

#define CH_CFG_USE_MEMCORE TRUE

Core Memory Manager APIs.

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

**Note**

The default is TRUE.
6.8 Options

6.8.2.27 CH_CFG_MEMCORE_SIZE

#define CH_CFG_MEMCORE_SIZE 0

Managed RAM size.

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

Note

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

Requires CH_CFG_USE_MEMCORE.

6.8.2.28 CH_CFG_USE_HEAP

#define CH_CFG_USE_HEAP TRUE

Heap Allocator APIs.

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

Note

The default is TRUE.

Requires CH_CFG_USE_MEMCORE and either CH_CFG_USE_MUTEXES or CH_CFG_USE_SEMAPHORE.

Mutexes are recommended.

6.8.2.29 CH_CFG_USE_MEMPOOLS

#define CH_CFG_USE_MEMPOOLS TRUE

Memory Pools Allocator APIs.

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

Note

The default is TRUE.
6.8.2.30 CH_CFG_USE_OBJ_FIFOS

#define CH_CFG_USE_OBJ_FIFOS TRUE

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

Note
The default is TRUE.

6.8.2.31 CH_CFG_USE_PIPES

#define CH_CFG_USE_PIPES TRUE

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

Note
The default is TRUE.

6.8.2.32 CH_CFG_USE_OBJ_CACHES

#define CH_CFG_USE_OBJ_CACHES TRUE

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

Note
The default is TRUE.

6.8.2.33 CH_CFG_USE_DELEGATES

#define CH_CFG_USE_DELEGATES TRUE

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

Note
The default is TRUE.
6.8 Options

6.8.2.34 CH_CFG_USE_JOBS

#define CH_CFG_USE_JOBS TRUE

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

Note
The default is TRUE.

6.8.2.35 CH_CFG_USE_FACTORY

#define CH_CFG_USE_FACTORY TRUE

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

Note
The default is FALSE.

6.8.2.36 CH_CFG_FACTORY_MAX_NAMES_LENGTH

#define CH_CFG_FACTORY_MAX_NAMES_LENGTH 8

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

6.8.2.37 CH_CFG_FACTORY_OBJECTS_REGISTRY

#define CH_CFG_FACTORY_OBJECTS_REGISTRY TRUE

Enables the registry of generic objects.

6.8.2.38 CH_CFG_FACTORY_GENERIC_BUFFERS

#define CH_CFG_FACTORY_GENERIC_BUFFERS TRUE

Enables factory for generic buffers.
6.8.2.39 CH_CFG_FACTORY_SEMAPHORES

#define CH_CFG_FACTORY_SEMAPHORES TRUE

Enables factory for semaphores.

6.8.2.40 CH_CFG_FACTORY_MAILBOXES

#define CH_CFG_FACTORY_MAILBOXES TRUE

Enables factory for mailboxes.

6.8.2.41 CH_CFG_FACTORY_OBJ_FIFOS

#define CH_CFG_FACTORY_OBJ_FIFOS TRUE

Enables factory for objects FIFOs.

6.8.2.42 CH_CFG_FACTORY_PIPES

#define CH_CFG_FACTORY_PIPES TRUE

Enables factory for Pipes.

6.8.2.43 CH_DBG_STATISTICS

#define CH_DBG_STATISTICS FALSE

Debug option, kernel statistics.

Note

The default is FALSE.
6.8.2.44 CH_DBG_SYSTEM_STATE_CHECK

#define CH_DBG_SYSTEM_STATE_CHECK TRUE

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

Note
The default is FALSE.

6.8.2.45 CH_DBG_ENABLE_CHECKS

#define CH_DBG_ENABLE_CHECKS TRUE

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

Note
The default is FALSE.

6.8.2.46 CH_DBG_ENABLE_ASSERTS

#define CH_DBG_ENABLE_ASSERTS TRUE

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

Note
The default is FALSE.

6.8.2.47 CH_DBG_TRACE_MASK

#define CH_DBG_TRACE_MASK CH_DBG_TRACE_MASK_ALL

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

Note
The default is CH_DBG_TRACE_MASK_DISABLED.
6.8.2.48 CH_DBG_TRACE_BUFFER_SIZE

#define CH_DBG_TRACE_BUFFER_SIZE 128

Trace buffer entries.

Note
The trace buffer is only allocated if CH_DBG_TRACE_MASK is different from CH_DBG_TRACE_MASK_DDISABLED.

6.8.2.49 CH_DBG_ENABLE_STACK_CHECK

#define CH_DBG_ENABLE_STACK_CHECK TRUE

Debug option, stack checks.

If enabled then a runtime stack check is performed.

Note
The default is FALSE.
The stack check is performed in a architecture/port dependent way. It may not be implemented or some ports.
The default failure mode is to halt the system with the global panic_msg variable set to NULL.

6.8.2.50 CH_DBG_FILL_THREADS

#define CH_DBG_FILL_THREADS TRUE

Debug option, stacks initialization.

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

Note
The default is FALSE.
### 6.8.2.51 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.

### 6.8.2.52 CH_CFG_SYSTEM_EXTRA_FIELDS

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

System structure extension.

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

### 6.8.2.53 CH_CFG_SYSTEM_INIT_HOOK

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

**Value:**

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

System initialization hook.

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

### 6.8.2.54 CH_CFG_OS_INSTANCE_EXTRA_FIELDS

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

OS instance structure extension.

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

### 6.8.2.55 CH_CFG_OS_INSTANCE_INIT_HOOK

```c
#define CH_CFG_OS_INSTANCE_INIT_HOOK( oip )
```

**Value:**

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

OS instance initialization hook.
Parameters

| in | oip | pointer to the os_instance_t structure |

### 6.8.2.56 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.

### 6.8.2.57 CH_CFG_THREAD_INIT_HOOK

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

**Value:**

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

Threads initialization hook.

User initialization code added to the _thread_init() function.

**Note**

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

Parameters

| in | tp | pointer to the thread_t structure |

### 6.8.2.58 CH_CFG_THREAD_EXIT_HOOK

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

**Value:**

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

Threads finalization hook.

User finalization code added to the chThdExit() API.
6.8 Options

Parameters

| in | tp | pointer to the thread_t structure |

6.8.2.59 CH_CFG_CONTEXT_SWITCH_HOOK

#define CH_CFG_CONTEXT_SWITCH_HOOK(
    ntp,
    otp )

Value:

    { /* Context switch code here.*/
    }

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

Parameters

| in | ntp | thread being switched in |
| in | otp | thread being switched out |

6.8.2.60 CH_CFG_IRQ_PROLOGUE_HOOK

#define CH_CFG_IRQ_PROLOGUE_HOOK( )

Value:

    { /* IRQ prologue code here.*/
    }

ISR enter hook.

6.8.2.61 CH_CFG_IRQ_EPILOGUE_HOOK

#define CH_CFG_IRQ_EPILOGUE_HOOK( )

Value:

    { /* IRQ epilogue code here.*/
    }

ISR exit hook.
6.8.2.62  CH_CFG_IDLE_ENTER_HOOK

#define CH_CFG_IDLE_ENTER_HOOK( )

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

Idle thread enter hook.

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

6.8.2.63  CH_CFG_IDLE_LEAVE_HOOK

#define CH_CFG_IDLE_LEAVE_HOOK( )

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

Idle thread leave hook.

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

6.8.2.64  CH_CFG_IDLE_LOOP_HOOK

#define CH_CFG_IDLE_LOOP_HOOK( )

Value:
{ 
    /* Idle loop code here.*/
}

Idle Loop hook.

This hook is continuously invoked by the idle thread loop.
6.8.2.65 **CH_CFG_SYSTEM_TICK_HOOK**

```c
#define CH_CFG_SYSTEM_TICK_HOOK( )
```

**Value:**

```c
/* 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.

6.8.2.66 **CH_CFG_SYSTEM_HALT_HOOK**

```c
#define CH_CFG_SYSTEM_HALT_HOOK( reason )
```

**Value:**

```c
/* System halt code here. */
```

System halt hook.

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

6.8.2.67 **CH_CFG_TRACE_HOOK**

```c
#define CH_CFG_TRACE_HOOK( tep )
```

**Value:**

```c
/* Trace code here. */
```

Trace hook.

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

6.8.2.68 **CH_CFG_RUNTIME_FAULTS_HOOK**

```c
#define CH_CFG_RUNTIME_FAULTS_HOOK( mask )
```

**Value:**

```c
/* Faults handling code here. */
```

Runtime Faults Collection Unit hook.

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

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

6.10.1 Detailed Description

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

6.11.1 Detailed Description

Modules

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

6.12.1 Detailed Description

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

Note

This module does not export any functionality.
6.13 OS Types and Structures

6.13.1 Detailed Description

Macros

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

Kernel types

- `typedef port_rtcnt_t rtcnt_t`
- `typedef port_rttme_t rttime_t`
- `typedef port_syssts_t syssts_t`
- `typedef port_stkalign_t stkalign_t`
- `typedef uint8_t tmode_t`
- `typedef uint8_t tstate_t`
- `typedef uint8_t trefs_t`
- `typedef uint8_t tslices_t`
- `typedef uint32_t tmode_t`
- `typedef uint32_t tsprio_t`
- `typedef int32_t msg_t`
- `typedef int32_t eventid_t`
- `typedef uint32_t eventmask_t`
- `typedef uint32_t eventflags_t`
- `typedef int32_t cnt_t`
- `typedef uint32_t ucnt_t`

Typedefs

- `typedef unsigned core_id_t`
  Type of a core identifier.
- `typedef struct ch_thread thread_t`
  Type of a thread structure.
- `typedef struct ch_os_instance os_instance_t`
  Type of an OS instance structure.
- `typedef struct ch_virtual_timer virtual_timer_t`
  Type of a Virtual Timer.
- `typedef void(vtfunc_t) (virtual_timer_t *vtp, void *p)`
  Type of a Virtual Timer callback function.
- `typedef struct ch_virtual_timers_list virtual_timers_list_t`
6.13 OS Types and Structures

Type of virtual timers list header.
• typedef struct ch_registry registry_t
  Type of a registry structure.
• typedef thread_t * thread_reference_t
  Type of a thread reference.
• typedef struct ch_threads_queue threads_queue_t
  Type of a threads queue.
• typedef struct ch_ready_list ready_list_t
  Type of a ready list header.
• typedef struct ch_os_instance_config os_instance_config_t
  Type of an system instance configuration.
• typedef struct ch_system ch_system_t
  Type of system data structure.

Data Structures

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

Functions

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

Enumerations

• enum system_state_t
  Global state of the operating system.

6.13.2 Macro Definition Documentation
6.13.2.1 __CH_STRINGIFY

#define __CH_STRINGIFY(
     a ) #a

Utility to make the parameter a quoted string.
6.13 OS Types and Structures

6.13.2.2 __CH_OFFSETOF

#define __CH_OFFSETOF( st, m )

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

Structure field offset utility.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>st</th>
<th>structured type name</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>m</td>
<td>field name in the structured type</td>
</tr>
</tbody>
</table>

Returns
The offset of the field in the structured type.

6.13.2.3 __CH_USED

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

Marks an expression result as used.

Parameters

| in  | x   | a valid expression |

6.13.2.4 likely

#define likely( x ) PORT_LIKELY(x)

Marks a boolean expression as likely true.

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

**Parameters**

| in | x | a valid expression |

**6.13.2.5 unlikely**

#define unlikely(
   x ) PORT_UNLIKELY(x)

Marks a boolean expression as likely false.

Note

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

**Parameters**

| in | x | a valid expression |

**6.13.2.6 threadref**

#define threadref(
   p ) ((thread_t ∗)(void ∗)(p))

Safe cast of a queue pointer to a thread pointer.

Note

Casting to a thread pointer should always be performed using this macro. Casting to threads pointer is allowed by design and this is the single check point for this operation.

**Parameters**

| in | p | pointer to a queue/list structure |

**Returns**

The pointer to the thread containing the queue/list element.
6.13 OS Types and Structures

6.13.3 Typedef Documentation

6.13.3.1 rtcnt_t

typedef port_rtcnt_t rtcnt_t

Realtime counter.

6.13.3.2 rttime_t

typedef port_rttime_t rttime_t

Realtime accumulator.

6.13.3.3 syssts_t

typedef port_syssts_t syssts_t

System status word.

6.13.3.4 stkalign_t

typedef port_stkalign_t stkalign_t

Stack alignment type.

6.13.3.5 tmode_t

typedef uint8_t tmode_t

Thread flags.

6.13.3.6 tstate_t

typedef uint8_t tstate_t

Thread state.
6.13.3.7  trefs_t

typedef uint8_t trefs_t

Thread references counter.

6.13.3.8  tslices_t

typedef uint8_t tslices_t

Thread time slices counter.

6.13.3.9  tprio_t

typedef uint32_t tprio_t

Thread priority.

6.13.3.10  msg_t

typedef int32_t msg_t

Inter-thread message.

6.13.3.11  eventid_t

typedef int32_t eventid_t

Numeric event identifier.

6.13.3.12  eventmask_t

typedef uint32_t eventmask_t

Mask of event identifiers.

6.13.3.13  eventflags_t

typedef uint32_t eventflags_t

Mask of event flags.
6.13.3.14 cnt_t

typedef int32_t cnt_t

Generic signed counter.

6.13.3.15 ucnt_t

typedef uint32_t ucnt_t

Generic unsigned counter.

6.13.3.16 core_id_t

typedef unsigned core_id_t

Type of a core identifier.

Note

Core identifiers have ranges from 0 to PORT_CORES_NUMBER - 1.

6.13.3.17 thread_t

typedef struct ch_thread thread_t

Type of a thread structure.

6.13.3.18 os_instance_t

typedef struct ch_os_instance os_instance_t

Type of an OS instance structure.

6.13.3.19 virtual_timer_t

typedef struct ch_virtual_timer virtual_timer_t

Type of a Virtual Timer.

6.13.3.20 vfunc_t

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

Type of a Virtual Timer callback function.
Module Documentation

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>vtp</th>
<th>pointer to the virtual_timer_t calling this callback</th>
</tr>
</thead>
<tbody>
<tr>
<td>[in]</td>
<td>p</td>
<td>optional argument to the callback</td>
</tr>
</tbody>
</table>

Returns

The interval to be reloaded into the timer or zero.

Return values

| 0  | if the timer must not be reloaded. |

6.13.3.21 virtual_timers_list_t

typedef struct ch_virtual_timers_list virtual_timers_list_t

Type of virtual timers list header.

Note

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

6.13.3.22 registry_t

typedef struct ch_registry registry_t

Type of a registry structure.

6.13.3.23 thread_reference_t

typedef thread_t* thread_reference_t

Type of a thread reference.
6.13.3.24 threads_queue_t

typedef struct ch_threads_queue threads_queue_t
Type of a threads queue.

6.13.3.25 ready_list_t

typedef struct ch_ready_list ready_list_t
Type of a ready list header.

6.13.3.26 os_instance_config_t

typedef struct ch_os_instance_config os_instance_config_t
Type of an system instance configuration.

6.13.3.27 ch_system_t

typedef struct ch_system ch_system_t
Type of system data structure.

6.13.4 Enumeration Type Documentation

6.13.4.1 system_state_t

enum system_state_t
Global state of the operating system.

6.13.5 Function Documentation

6.13.5.1 chSysHalt()

void chSysHalt ( const char * reason )
Halts the system.
This function is invoked by the operating system when an unrecoverable error is detected, for example because a programming error in the application code that triggers an assertion while in debug mode.

Note
Can be invoked from any system state.
Parameters

| in  | 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:
6.14 OS Instances

6.14.1 Detailed Description

OS instances management.

Macros

- **#define __instance_get_currthread(oip) (oip)->rlist.current**
  
  Current thread pointer get macro.

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

Functions

- **static void __idle_thread (void *p)**
  
  This function implements the idle thread infinite loop.

- **void chInstanceObjectInit (os_instance_t *oip, const os_instance_config_t *oicp)**
  
  Initializes a system instance.

6.14.2 Macro Definition Documentation

6.14.2.1 __instance_get_currthread

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

Current thread pointer get macro.

Note

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

6.14.2.2 __instance_set_currthread

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

Current thread pointer set macro.
6.14.3 Function Documentation

6.14.3.1 __idle_thread()

```c
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

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

6.14.3.2 chInstanceObjectInit()

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

Initializes a system instance.

Note

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

Parameters

<table>
<thead>
<tr>
<th>out</th>
<th>oip</th>
<th>pointer to the os_instance_t structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>oicp</td>
<td>pointer to the os_instance_config_t structure</td>
</tr>
</tbody>
</table>

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

Here is the call graph for this function:
6.15 Runtime Faults Collection Unit

6.15.1 Detailed Description

Runtime Faults Collection Unit service.

Macros

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

Predefined Faults

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

Typedefs

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

Data Structures

- `struct ch_rfcu`
  
  Type of an RFCU structure.

Functions

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

6.15.2 Macro Definition Documentation

6.15.2.1 CH_RFCU_ALL_FAULTS

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

Mask of all faults.
6.15 Runtime Faults Collection Unit

6.15.3 Typedef Documentation

6.15.3.1 rfcu_mask_t

typedef uint32_t rfcu_mask_t

Type of a faults mask.

6.15.3.2 rfcu_t

typedef struct ch_rfcu rfcu_t

Type of an RFCU structure.

6.15.4 Function Documentation

6.15.4.1 chRFCUCollectFaultsI()

void chRFCUCollectFaultsI (  
    rfcu_mask_t mask )

Adds fault flags to the current mask.

Parameters

<table>
<thead>
<tr>
<th></th>
<th>mask</th>
<th>fault flags to be added</th>
</tr>
</thead>
</table>

6.15.4.2 chRFCUGetAndClearFaultsI()

rfcu_mask_t chRFCUGetAndClearFaultsI (  
    rfcu_mask_t mask )

Returns the current faults mask clearing it.

Parameters

<table>
<thead>
<tr>
<th></th>
<th>mask</th>
<th>mask of faults to be read and cleared</th>
</tr>
</thead>
</table>
Returns

The current faults mask.

Returns values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>If no faults were collected since last call to this function.</td>
</tr>
</tbody>
</table>

6.15.4.3 __rfcu_object_init()

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

Runtime Faults Collection Unit initialization.

Note

Internal use only.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>out rfcup</td>
<td>Pointer to the rfcu_t structure</td>
</tr>
</tbody>
</table>

Function Class:

Not an API, this function is for internal use only.
6.16 Lists and Queues

6.16.1 Detailed Description

Macros

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

  Static queue object initializer.

Typedefs

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

Data Structures

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

Functions

- `static void ch_list_init (ch_list_t *lp)`

  List initialization.
- `static bool ch_list_isempty (ch_list_t *lp)`

  Evaluates to `true` if the specified list is empty.
- `static bool ch_list_notempty (ch_list_t *lp)`

  Evaluates to `true` if the specified list is not empty.
- `static void ch_list_link (ch_list_t *lp, ch_list_t *p)`

  Pushes an element on top of a stack list.
- `static ch_list_t * ch_list_unlink (ch_list_t *lp)`

  Pops an element from the top of a stack list and returns it.
- `static void ch_queue_init (ch_queue_t *q)`

  Queue initialization.
- `static bool ch_queue_isempty (const ch_queue_t *q)`
Evaluates to true if the specified queue is empty.

- static bool ch_queue_notempty (const ch_queue_t *qp)

Evaluates to true if the specified queue is not empty.

- static void ch_queue_insert (ch_queue_t *qp, ch_queue_t *p)
  Inserts an element into a queue.

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

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

- static ch_queue_t * ch_queue_dequeue (ch_queue_t *p)
  Removes an element from a queue and returns it.

- static void ch_pqueue_init (ch_priority_queue_t *pqp)
  Priority queue initialization.

- static ch_priority_queue_t * ch_pqueue_remove_highest (ch_priority_queue_t *pqp)
  Removes the highest priority element from a priority queue and returns it.

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

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

- static void ch_dlist_init (ch_delta_list_t *dlhp)
  Delta list initialization.

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

- static bool ch_dlist_notempty (ch_delta_list_t *dlhp)
  Evaluates to true if the specified queue is not empty.

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

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

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

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

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

- static ch_delta_list_t * ch_dlist_remove_first (ch_delta_list_t *dlhp)
  Dequeues an element from the delta list.

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

6.16.2 Macro Definition Documentation

6.16.2.1 __CH_QUEUE_DATA

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

Data part of a static queue object initializer.

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

Parameters

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

6.16.2.2 CH_QUEUE_DECL

```c
#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 |

6.16.3 Typedef Documentation

6.16.3.1 ch_list_t

```c
typedef struct ch_list ch_list_t
```

Type of a generic single link list header and element.

6.16.3.2 ch_queue_t

```c
typedef struct ch_queue ch_queue_t
```

Type of a generic bidirectional linked list header and element.

6.16.3.3 ch_priority_queue_t

```c
typedef struct ch_priority_queue ch_priority_queue_t
```

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

typedef struct ch_delta_list ch_delta_list_t

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

6.16.4 Function Documentation

6.16.4.1 ch_list_init()

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

List initialization.

Parameters

out lp pointer to the list header

Function Class:

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

6.16.4.2 ch_list_isempty()

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

Evaluates to true if the specified list is empty.

Parameters

in lp pointer to the list header

Returns

The status of the list.

Function Class:

Not an API, this function is for internal use only.
6.16.3  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.

6.16.4  ch_list_link()

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

Pushes an element on top of a stack list.

Parameters

| in | lp | the pointer to the list header |
| in | p  | the pointer to the element to be inserted in the list |

Function Class:

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

6.16.5  ch_list_unlink()

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

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

Precondition

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

\textbf{in} \quad \textbf{lp} \quad \text{the pointer to the list header}

Returns

The removed element pointer.

Function Class:

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

\textbf{6.16.4.6} \ \texttt{ch\_queue\_init()}

\begin{verbatim}
static void ch_queue_init (  
    ch_queue_t * qp ) [inline], [static]
\end{verbatim}

Queue initialization.

Parameters

\textbf{out} \quad \textbf{qp} \quad \text{pointer to the queue header}

Function Class:

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

\textbf{6.16.4.7} \ \texttt{ch\_queue\_isempty()}

\begin{verbatim}
static bool ch_queue_isempty (  
    const ch_queue_t * qp ) [inline], [static]
\end{verbatim}

Evaluates to \texttt{true} if the specified queue is empty.

Parameters

\textbf{in} \quad \textbf{qp} \quad \text{pointer to the queue header}

Returns

The status of the queue.
Function Class:
    Not an API, this function is for internal use only.

### 6.16.4.8 ch_queue_notempty()

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

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

**Parameters**

- `in` `qp` pointer to the queue header

**Returns**

The status of the queue.

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

### 6.16.4.9 ch_queue_insert()

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

Inserts an element into a queue.

**Parameters**

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

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

### 6.16.4.10 ch_queue_fifo_remove()

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

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

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

Parameters
in 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.

6.16.4.11 ch_queue_lifo_remove()

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

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

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

Parameters
in 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.
6.16.4.12  ch_queue_dequeue()

```c
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.

6.16.4.13  ch_pqueue_init()

```c
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.

6.16.4.14  ch_pqueue_remove_highest()

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

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

| in | 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.

6.16.4.15  ch_pqueue_insert_behind()

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

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

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

Parameters

| in | 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.

6.16.4.16  ch_pqueue_insert_ahead()

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

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

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

| in  | 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.

6.16.4.17  ch_dlist_init()

static void ch_dlist_init (  
    ch_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.

6.16.4.18  ch_dlist_isempty()

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

Evaluates to true if the specified delta list is empty.

Parameters

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

Returns

The status of the delta list.
6.16.4.19 ch_dlist_notempty()

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

Evaluates to true if the specified queue is not empty.

Parameters

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

Returns

The status of the delta list.

Function Class:

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

6.16.4.20 ch_dlist_islast()

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

Last element in the delta 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.

6.16.4.21 ch_dlist_isfirst()

static bool ch_dlist_isfirst (  }
Fist element in the delta list check.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in <code>dlhp</code></td>
<td>pointer to the delta list header</td>
</tr>
<tr>
<td>in <code>dlp</code></td>
<td>pointer to the delta list element</td>
</tr>
</tbody>
</table>

Function Class:

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

6.16.4.22  `ch_dlist_insert_after()`

static void ch_dlist_insert_after {
  ch_delta_list_t * dlhp,
  ch_delta_list_t * dlp,
  sysinterval_t delta } [inline], [static]

Inserts an element after another header element.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in <code>dlhp</code></td>
<td>pointer to the delta list header element</td>
</tr>
<tr>
<td>in <code>dlp</code></td>
<td>element to be inserted after the header element</td>
</tr>
<tr>
<td>in <code>delta</code></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.

6.16.4.23  `ch_dlist_insert_before()`

static void ch_dlist_insert_before {
  ch_delta_list_t * dlhp,
  ch_delta_list_t * dlp,
  sysinterval_t delta } [inline], [static]

Inserts an element before another header element.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in <code>dlhp</code></td>
<td>pointer to the delta list header element</td>
</tr>
<tr>
<td>in <code>dlp</code></td>
<td>element to be inserted before the header element</td>
</tr>
<tr>
<td>in <code>delta</code></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.

6.16.4.24  ch_dlist_insert()

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

Inserts an element in a delta list.

Parameters

<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:

```
ch_dlist_insert  ch_dlist_insert_before
```

6.16.4.25  ch_dlist_remove_first()

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

Dequeues an element from the delta list.

Parameters

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

ChibiOS/RT
6.16 Lists and Queues

Function Class:

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

6.16.4.26  ch_dlist_dequeue()

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

Dequeues an element from the delta list.

Parameters

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

Function Class:

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

6.17.1 Detailed Description

This module provides the default portable scheduler code.

Macros

- `#define firstprio(rlp) ((rlp)->next->prio)`
  Returns the priority of the first thread on the given ready list.

- `#define __sch_get_currthread() __instance_get_currthread(currcore)`
  Current thread pointer get macro.

Wakeup status codes

- `#define MSG_OK (msg_t)0`
  Normal wakeup message.

- `#define MSG_TIMEOUT (msg_t)-1`
  Wakeup caused by a timeout condition.

- `#define MSG_RESET (msg_t)-2`
  Wakeup caused by a reset condition.

Priority constants

- `#define NOPRIO (tprio_t)0`
  Ready list header priority.

- `#define IDLEPRIO (tprio_t)1`
  Idle priority.

- `#define LOWPRIO (tprio_t)2`
  Lowest priority.

- `#define NORMALPRIO (tprio_t)128`
  Normal priority.

- `#define HIGHPRIO (tprio_t)255`
  Highest priority.
Thread states

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Functions

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

6.17.2 Macro Definition Documentation

6.17.2.1 MSG_OK

#define MSG_OK (msg_t)0

Normal wakeup message.

6.17.2.2 MSG_TIMEOUT

#define MSG_TIMEOUT (msg_t)-1

Wakeup caused by a timeout condition.

6.17.2.3 MSG_RESET

#define MSG_RESET (msg_t)-2

Wakeup caused by a reset condition.

6.17.2.4 NOPRIO

#define NOPRIO (tprio_t)0

Ready list header priority.

6.17.2.5 IDLEPRIO

#define IDLEPRIO (tprio_t)1

Idle priority.
6.17.2.6 LOWPRIO

#define LOWPRIO (tprio_t)2

Lowest priority.

6.17.2.7 NORMALPRIO

#define NORMALPRIO (tprio_t)128

Normal priority.

6.17.2.8 HIGHPRIO

#define HIGHPRIO (tprio_t)255

Highest priority.

6.17.2.9 CH_STATE_READY

#define CH_STATE_READY (tstate_t)0

Waiting on the ready list.

6.17.2.10 CH_STATE_CURRENT

#define CH_STATE_CURRENT (tstate_t)1

Currently running.

6.17.2.11 CH_STATE_WTSTART

#define CH_STATE_WTSTART (tstate_t)2

Just created.
6.17 Scheduler

6.17.2.12 CH_STATE_SUSPENDED

#define CH_STATE_SUSPENDED (tstate_t)3

Suspended state.

6.17.2.13 CH_STATE_QUEUED

#define CH_STATE_QUEUED (tstate_t)4

On a queue.

6.17.2.14 CH_STATE_WTSEM

#define CH_STATE_WTSEM (tstate_t)5

On a semaphore.

6.17.2.15 CH_STATE_WTMTX

#define CH_STATE_WTMTX (tstate_t)6

On a mutex.

6.17.2.16 CH_STATE_WTCOND

#define CH_STATE_WTCOND (tstate_t)7

On a cond.variable.

6.17.2.17 CH_STATE_SLEEPING

#define CH_STATE_SLEEPING (tstate_t)8

Sleeping.
6.17.2.18 CH_STATE_WTEXIT

#define CH_STATE_WTEXIT (tstate_t)9
Waiting a thread.

6.17.2.19 CH_STATE_WTOREVT

#define CH_STATE_WTOREVT (tstate_t)10
One event.

6.17.2.20 CH_STATE_WTANDEVT

#define CH_STATE_WTANDEVT (tstate_t)11
Several events.

6.17.2.21 CH_STATE_SNDMSGQ

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

6.17.2.22 CH_STATE_SNDMSG

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

6.17.2.23 CH_STATE_WTMSG

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

#define CH_STATE_FINAL (tstate_t)15

Thread terminated.

6.17.25  CH_STATE_NAMES

#define CH_STATE_NAMES

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

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

6.17.26  CH_FLAG_MODE_MASK

#define CH_FLAG_MODE_MASK (tmode_t)3U

Thread memory mode mask.

6.17.27  CH_FLAG_MODE_STATIC

#define CH_FLAG_MODE_STATIC (tmode_t)0U

Static thread.

6.17.28  CH_FLAG_MODE_HEAP

#define CH_FLAG_MODE_HEAP (tmode_t)1U

Thread allocated from a Memory Heap.
6.17.2.29  CH_FLAG_MODE_MPOOL

#define CH_FLAG_MODE_MPOOL (tmode_t)2U

Thread allocated from a Memory Pool.

6.17.2.30  CH_FLAG_TERMINATE

#define CH_FLAG_TERMINATE (tmode_t)4U

Termination requested flag.

6.17.2.31  firstprio

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

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

Function Class:

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

6.17.2.32  __sch_get_curthread

#define __sch_get_curthread( ) __instance_get_curthread(currcore)

Current thread pointer get macro.

Note

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

6.17.3  Function Documentation
6.17 Scheduler

6.17.3.1 __sch_ready_behind()

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

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

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

**Precondition**

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

**Postcondition**

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

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

Returns

The thread pointer.

Function Class:

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

Here is the call graph for this function:

![Call Graph]

6.17.3.2 __sch_ready_ahead()

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

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

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

Precondition

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

Postcondition

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

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in tp</td>
<td>the thread to be made ready</td>
</tr>
</tbody>
</table>

Returns

The thread pointer.

Function Class:

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

Here is the call graph for this function:

6.17.3.3 __sch_reschedule_behind()

static void __sch_reschedule_behind ( void ) [static]

Switches to the first thread on the runnable queue.
The current thread is positioned in the ready list behind all threads having the same priority. The thread regains its time quantum.

Note

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

Function Class:

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

Here is the call graph for this function:
6.17.3.4  __sch_reschedule_ahead()

static void __sch_reschedule_ahead (  
  void ) [static]

Switches to the first thread on the runnable queue.

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

Note

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

Function Class:

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

Here is the call graph for this function:

<table>
<thead>
<tr>
<th>Function Call</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>__sch_reschedule_ahead</td>
<td></td>
</tr>
<tr>
<td>__sch_ready_ahead</td>
<td></td>
</tr>
<tr>
<td>ch_pqueue_remove_highest</td>
<td></td>
</tr>
<tr>
<td>__trace_ready</td>
<td></td>
</tr>
<tr>
<td>ch_pqueue_insert_ahead</td>
<td></td>
</tr>
</tbody>
</table>

6.17.3.5  ch_sch_prio_insert()

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

Inserts a thread into a priority ordered queue.

Note

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

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in qp</td>
<td>the pointer to the threads list header</td>
</tr>
<tr>
<td>in tp</td>
<td>the pointer to the thread to be inserted in the list</td>
</tr>
</tbody>
</table>
Function Class:
Not an API, this function is for internal use only.

6.17.3.6 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
```
in tp the thread to be made ready
```

Returns
The thread pointer.
Module Documentation

Function Class:

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

Here is the call graph for this function:

![Call Graph]

6.17.3.7 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
6.17 Scheduler

Function Class:

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

Here is the call graph for this function:

![Call Graph](image)

6.17.3.8 chSchGoSleepTimeoutS()

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

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

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

**Parameters**

<table>
<thead>
<tr>
<th></th>
<th><strong>newstate</strong></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>
<tr>
<td></td>
<td></td>
<td>• TIME_INFINITE the thread enters an infinite sleep state, this is equivalent to invoking chSchGoSleepS() but, of course, less efficient.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• TIME_IMMEDIATE this value is not allowed.</td>
</tr>
</tbody>
</table>

**Returns**

The wakeup message.

**Return values**

```c
MSG_TIMEOUT  if a timeout occurs.
```
Function Class:

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

Here is the call graph for this function:

![Call Graph Image]

### 6.17.3.9 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>in</td>
<td><code>ntp</code></td>
</tr>
<tr>
<td>in</td>
<td><code>msg</code></td>
</tr>
</tbody>
</table>
6.17 Scheduler

Function Class:

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

Here is the call graph for this function:

![Call Graph](image)

6.17.3.10 chSchRescheduleS()

```c
void chSchRescheduleS (  
                      )
```

Performs a reschedule if a higher priority thread is runnable.

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

Note:

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

Function Class:

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

Here is the call graph for this function:

![Call Graph](image)
6.17.3.11 chSchIsPreemptionRequired()

bool chSchIsPreemptionRequired (  
        void )  
Evaluates if preemption is required.

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

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

Return values

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

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

6.17.3.12 chSchDoPreemption()

void chSchDoPreemption (  
        void )  
Switches to the first thread on the runnable queue.

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

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

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

Here is the call graph for this function:
6.17.3.13  chSchPreemption()

void chSchPreemption ( 
    void )

All-in-one preemption code.

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

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

Here is the call graph for this function:

6.17.3.14  chSchDoYieldS()

void chSchDoYieldS ( 
    void )

Yields the time slot.

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

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

Here is the call graph for this function:

6.17.3.15 chSchSelectFirst()

thread_t * chSchSelectFirst ( 
        void )

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

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

Note

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

Returns

The pointer to the thread being switched in.

Function Class:

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

Here is the call graph for this function:
6.18 Base Kernel Services

6.18.1 Detailed Description

Modules

- System Management
- Memory Alignment
- Time and Intervals
- Virtual Timers
- Threads
- Time Measurement
6.19  System Management

6.19.1  Detailed Description

System related APIs and services:

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

Macros

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

Masks of executable integrity checks.

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

ISRs abstraction macros

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

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

Time conversion utilities for the realtime counter

- `#define S2RTC(freq, sec) ((freq) * (sec))`
  
  Seconds to realtime counter.

- `#define MS2RTC(freq, msec) (rtcnt_t)(((freq) + 999UL) / 1000UL) * (msec)`
  
  Milliseconds to realtime counter.

- `#define US2RTC(freq, usec) (rtcnt_t)(((freq) + 999999UL) / 1000000UL) * (usec)`
  
  Microseconds to realtime counter.

- `#define RTC2S(freq, n) (((n) - 1UL) / (freq)) + 1UL`
  
  Realtime counter cycles to seconds.

- `#define RTC2MS(freq, n) (((n) - 1UL) / ((freq) / 1000UL)) + 1UL`
  
  Realtime counter cycles to milliseconds.

- `#define RTC2US(freq, n) (((n) - 1UL) / ((freq) / 1000000UL)) + 1UL`
  
  Realtime counter cycles to microseconds.

Functions

- `static CH_SYS_CORE0_MEMORY THD_WORKING_AREA (ch_c0_idle_thread_wa, PORT_IDLE_THRE→ AD_STACK_SIZE)`
  
  Working area for core 0 idle thread.

- `static CH_SYS_CORE1_MEMORY THD_WORKING_AREA (ch_c1_idle_thread_wa, PORT_IDLE_THRE→ AD_STACK_SIZE)`
  
  Working area for core 1 idle thread.

- `void chSysWaitSystemState (system_state_t state)`
  
  Waits for the system state to be equal to the specified one.

- `void chSysInit (void)`
  
  System initialization.

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

- `thread_t ∗chSysGetIdleThreadX (void)`
  
  Returns a pointer to the idle thread.

- `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 void chSysNotifyInstance (os_instance_t *oip)
  
  *Notifies an OS instance to check for reschedule.*

**Variables**

• ch_system_t ch_system
  
  *System root object.*
• CH_SYS_CORE0_MEMORY os_instance_t ch0
  
  *Core 0 OS instance.*
• const os_instance_config_t ch_core0_cfg
  
  *Core 0 OS instance configuration.*
• CH_SYS_CORE1_MEMORY os_instance_t ch1
  
  *Core 1 OS instance.*
• const os_instance_config_t ch_core1_cfg
  
  *Core 1 OS instance configuration.*

### 6.19.2 Macro Definition Documentation

**6.19.2.1 CH_SYS_CORE0_MEMORY**

#define CH_SYS_CORE0_MEMORY PORT_CORE0_BSS_SECTION

*Core zero memory affinity macro.*

**Note**

The memory is meant to be reachable by both cores but preferred by core zero.

Only uninitialized variables can be tagged with this attribute.
6.19.2.2  CH_SYS_CORE1_MEMORY

#define CH_SYS_CORE1_MEMORY PORT_CORE1_BSS_SECTION

Core one memory affinity macro.

Note

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

6.19.2.3  currcore

#define currcore ch_system.instances[port_get_core_id()]

Access to current core's instance structure.

6.19.2.4  CH_IRQ_IS_VALID_PRIORITY

#define CH_IRQ_IS_VALID_PRIORITY(    prio ) PORT_IRQ_IS_VALID_PRIORITY(prio)

Priority level validation macro.

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

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>prio</th>
<th>the priority level</th>
</tr>
</thead>
</table>

Returns

Priority range result.

Return values

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

6.19.2.5  CH_IRQ_IS_VALID_KERNEL_PRIORITY

#define CH_IRQ_IS_VALID_KERNEL_PRIORITY( |
Priority level validation macro.

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

Parameters

- **in prio**: the priority level

Returns

Priority range result.

Return values

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

### 6.19.2.6 CH_IRQ_PROLOGUE

#define CH_IRQ_PROLOGUE()

**Value:**

- PORT_IRQ_PROLOGUE();
- CH_CFG_IRQ_PROLOGUE_HOOK();
- __stats_increase_irq();
- __trace_isr_enter(__func__);
- __dbg_check_enter_isr()

IRQ handler enter code.

**Note**

Usually IRQ handlers functions are also declared naked.

On some architectures this macro can be empty.

**Function Class:**

Special function, this function has special requirements see the notes.
### 6.19.2.7 CH_IRQ_EPILOGUE

```c
#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.

---

### 6.19.2.8 CH_IRQ_HANDLER

```c
#define CH_IRQ_HANDLER( id ) PORT_IRQ_HANDLER(id)
```

Standard normal IRQ handler declaration.

**Note**

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

**Function Class:**

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

---

### 6.19.2.9 CH_FAST_IRQ_HANDLER

```c
#define CH_FAST_IRQ_HANDLER( id ) PORT_FAST_IRQ_HANDLER(id)
```

Standard fast IRQ handler declaration.

**Note**

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

**Function Class:**

Special function, this function has special requirements see the notes.
6.19.2.10 S2RTC

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

Seconds to realtime counter.

Converts from seconds to realtime counter cycles.

Note
The macro assumes that \(freq \geq 1\).

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.

6.19.2.11 MS2RTC

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

Milliseconds to realtime counter.

Converts from milliseconds to realtime counter cycles.

Note
The result is rounded upward to the next millisecond boundary.
The macro assumes that \(freq \geq 1000\).

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>
6.19 System Management

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.

6.19.2.12 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 \( freq \geq 1000000 \).

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>usec</td>
<td>number of microseconds</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.

6.19.2.13 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 \( freq \geq 1 \).
Parameters

\[
\begin{array}{|c|c|}
\hline
\text{in} & \text{freq} & \text{clock frequency, in Hz, of the realtime counter} \\
\text{in} & n & \text{number of cycles} \\
\hline
\end{array}
\]

Returns

The number of seconds.

Function Class:

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

6.19.2.14 RTC2MS

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

Realtime counter cycles to milliseconds.

Converts from realtime counter cycles number to milliseconds.

Note

The result is rounded up to the next millisecond boundary.

The macro assumes that \( \text{freq} \geq 1000 \).

Parameters

\[
\begin{array}{|c|c|}
\hline
\text{in} & \text{freq} & \text{clock frequency, in Hz, of the realtime counter} \\
\text{in} & n & \text{number of cycles} \\
\hline
\end{array}
\]

Returns

The number of milliseconds.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.
6.19.2.15 RTC2US

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

Realtime counter cycles to microseconds.

Converts from realtime counter cycles number to microseconds.

Note
The result is rounded up to the next microsecond boundary.
The macro assumes that freq >= 1000000.

Parameters

<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.

6.19.2.16 chSysGetRealtimeCounterX

#define chSysGetRealtimeCounterX() (rtcnt_t)port_rt_get_counter_value()

Returns the current value of the system real time counter.

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

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

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

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

Value:

\[
\text{\_\_\_trace\_switch(ntp, otp);} \\
\text{\_\_\_stats\_ctxswc(ntp, otp);} \\
\text{CH\_CFG\_CONTEXT\_SWITCH\_HOOK(ntp, otp);} \\
\text{port\_switch(ntp, otp);} \\
\]

Performs a context switch.

**Note**

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

**Parameters**

- **in ntp** the thread to be switched in
- **in otp** the thread to be switched out

**Function Class:**

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

### 6.19.3 Function Documentation

#### 6.19.3.1 THD_WORKING_AREA() [1/2]

```c
static CH\_SYS\_CORE0\_MEMORY THD\_WORKING\_AREA (
    ch\_c0\_idle\_thread\_wa ,
    PORT\_IDLE\_THREAD\_STACK\_SIZE ) [static]
```

Working area for core 0 idle thread.

#### 6.19.3.2 THD_WORKING_AREA() [2/2]

```c
static CH\_SYS\_CORE1\_MEMORY THD\_WORKING\_AREA (
    ch\_c1\_idle\_thread\_wa ,
    PORT\_IDLE\_THREAD\_STACK\_SIZE ) [static]
```

Working area for core 1 idle thread.
6.19.3.3 chSysWaitSystemState()

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

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

**Note**

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

**Function Class:**

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

6.19.3.4 chSysInit()

```c
void chSysInit ( void )
```

System initialization.

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

**Precondition**

Interrupts must disabled before invoking this function.

**Postcondition**

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

**Function Class:**

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

6.19.3.5 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:

6.19.3.6 chSysGetIdleThreadX()

```c
thread_t * chSysGetIdleThreadX ( void )
```

Returns a pointer to the idle thread.

Note

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

This function cannot be called from the idle thread itself, use chThdGetSelfX() in that case.

Returns

Pointer to the idle thread.

Function Class:

This is an X-Class API, this function can be invoked from any context.
6.19.3.7 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:
6.19.3.8 chSysTimerHandlerI()

void chSysTimerHandlerI ( 
    void )

Handles time ticks for round robin preemption and timer increments.

Decrement 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 \texttt{CH\_CFG\_TIME\_QUANTUM} macro, the round robin interval.

Function Class:

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

Here is the call graph for this function:
6.19.3.9 chSysGetStatusAndLockX()

syssts_t chSysGetStatusAndLockX ( void )

Returns the execution status and enters a critical zone.

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

Postcondition

The system is in a critical zone.

Returns

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

Function Class:

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

Here is the call graph for this function:

6.19.3.10 chSysRestoreStatusX()

void chSysRestoreStatusX ( syssts_t sts )

Restores the specified execution status and leaves a critical zone.

Note

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

Parameters

\textbf{in } \textit{sts} \textbf{the system status to be restored.}

Function Class:

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

Here is the call graph for this function:

6.19.3.11 \textbf{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 \texttt{PORT_SUPPORTS_RT}. 

ChibiOS/RT
6.19 System Management

Parameters

| in  | cnt       | the counter value to be tested |
| in  | start     | the start of the time window (inclusive) |
| in  | end       | the end of the time window (non inclusive) |

Return values

| true | current time within the specified time window. |
| false| current time not within the specified time window. |

Function Class:

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

6.19.3.12 chSysPolledDelayX()

void chSysPolledDelayX (rtcnt_t cycles)

Polled delay.

Note

The real delay is always few cycles in excess of the specified value.
This function is only available if the port layer supports the option PORT_SUPPORTS_RT.

Parameters

| in | 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:
6.19.3.13  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.

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

Function Class:

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

Here is the call graph for this function:

6.19.3.14  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() which could do more than just disable interrupts.
Function Class:
Special function, this function has special requirements see the notes.

Here is the call graph for this function:

6.19.3.15  chSysEnable()

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

Lowers the system interrupt priority mask to user level.

All the interrupt sources are enabled.

Note
Do not invoke this API from within a kernel lock.
This API is no replacement for chSysUnlock() which could do more than just enable interrupts.

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

Here is the call graph for this function:
6.19.3.16  

chSysLock()

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

Enters the kernel lock state.

Note

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

Function Class:

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

Here is the call graph for this function:

6.19.3.17  

chSysUnlock()

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

Leaves the kernel lock state.

Note

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

Function Class:

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

Here is the call graph for this function:
6.19 System Management

6.19.3.18 chSysLockFromISR()

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

Enters the kernel lock state from within an interrupt handler.

Note

This API may do nothing on some architectures, it is required because on ports that support preemptable 
interrupt handlers it is required to raise the interrupt mask to the same level of the system mutual exclusion 
zone.
It is good practice to invoke this API before invoking any I-class syscall from an interrupt handler.
The exact behavior of this function is port-dependent and could not be limited to disabling interrupts.
This API must be invoked exclusively from interrupt handlers.

Function Class:

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

Here is the call graph for this function:


6.19.3.19 chSysUnlockFromISR()

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

Leaves the kernel lock state from within an interrupt handler.

Note

This API may do nothing on some architectures, it is required because on ports that support preemptable 
interrupt handlers it is required to raise the interrupt mask to the same level of the system mutual exclusion 
zone.
It is good practice to invoke this API after invoking any I-class syscall from an interrupt handler.
The exact behavior of this function is port-dependent and could not be limited to enabling interrupts.
This API must be invoked exclusively from interrupt handlers.
Function Class:

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

Here is the call graph for this function:

![Call Graph](call_graph.png)

6.19.3.20 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:

![Call Graph](call_graph2.png)
6.19.3.21 chSysUnconditionalUnlock()

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

Unconditionally leaves the kernel lock state.

Note

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

Function Class:

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

Here is the call graph for this function:

6.19.3.22 chSysNotifyInstance()

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

Notifies an OS instance to check for reschedule.

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

Parameters

| in  | oip | pointer to the instance to be notified |

6.19.4 Variable Documentation
6.19.4.1 ch_system

ch_system_t ch_system

System root object.

6.19.4.2 ch0

CH_SYS_CORE0_MEMORY os_instance_t ch0

Core 0 OS instance.

6.19.4.3 ch_core0_cfg

const os_instance_config_t ch_core0_cfg

Initial value:

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

Core 0 OS instance configuration.

6.19.4.4 ch1

CH_SYS_CORE1_MEMORY os_instance_t ch1

Core 1 OS instance.

6.19.4.5 ch_core1_cfg

const os_instance_config_t ch_core1_cfg

Initial value:

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

Core 1 OS instance configuration.
6.20 Memory Alignment

6.20.1 Detailed Description

Memory Alignment services.

Memory alignment support macros

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

6.20.2 Macro Definition Documentation

6.20.2.1 MEM_ALIGN_MASK

`#define MEM_ALIGN_MASK(a) ((size_t)(a) - 1U)
Alignment mask constant.

Parameters

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

6.20.2.2 MEM_ALIGN_PREV

`#define MEM_ALIGN_PREV(p, a)
Value:
/
+ lint -save -e9033 [10.8] The cast is safe.*/
+ (size_t)(p) & ~MEM_ALIGN_MASK(a)
+ lint -restore*/
Aligns to the previous aligned memory address.
### 6.20.2.3 MEM_ALIGN_NEXT

```c
#define MEM_ALIGN_NEXT( 
  p, 
  a ) 
MEM_ALIGN_PREV((size_t)(p) + MEM_ALIGN_MASK(a), (a))
```

Aligns to the next aligned memory address.

#### Parameters

- **Variable to be aligned** 
  - **p**
- **Alignment** 
  - **a**
  - Must be a power of two

### 6.20.2.4 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

- **Variable to be aligned** 
  - **p**
- **Alignment** 
  - **a**
  - Must be a power of two

### 6.20.2.5 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 |
6.21 Time and Intervals

6.21.1 Detailed Description

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

Special time constants

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

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

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

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

Fast time conversion utilities

• #define TIME_S2I(secs) ((sysinterval_t)((time_conv_t)(secs) \* (time_conv_t)CH_CFG_ST_FREQUENCY))  
  Seconds to time interval.

• #define TIME_MS2I(msecs)  
  Milliseconds to time interval.

• #define TIME_US2I(usecs)  
  Microseconds to time interval.

• #define TIME_I2S(interval)  
  Time interval to seconds.

• #define TIME_I2MS(interval)  
  Time interval to milliseconds.

• #define TIME_I2US(interval)  
  Time interval to microseconds.

Secure time conversion utilities

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

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

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

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

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

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

• static systime_t chTimeAddX (systime_t systime, sysinterval_t interval)  
  Adds an interval to a system time returning a system time.
• static systime_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.
• static systimestamp_t chTimeStampAddX (systimestamp_t stamp, sysinterval_t interval)
  Adds an interval to a time stamp returning a time stamp.
• static sysinterval_t chTimeStampDiffX (systimestamp_t start, systimestamp_t end)
  Subtracts two time stamps returning an interval.
• static bool chTimestampInRangeX (systimestamp_t stamp, systimestamp_t start, systimestamp_t end)
  Checks if the specified time stamp is within the specified time stamps range.

Typedefs

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

6.21.2 Macro Definition Documentation

6.21.2.1 TIME_IMMEDIATE

#define TIME_IMMEDIATE ((sysinterval_t)0)

Zero interval specification for some functions with a timeout specification.

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

#define TIME_INFINITE ((sysinterval_t)-1)

Infinite interval specification for all functions with a timeout specification.

Note

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

6.21.2.3 TIME_MAX_INTERVAL

#define TIME_MAX_INTERVAL ((sysinterval_t)-2)

Maximum interval constant usable as timeout.

6.21.2.4 TIME_MAX_SYSTIME

#define TIME_MAX_SYSTIME ((systime_t)-1)

Maximum system of system time before it wraps.

6.21.2.5 TIME_S2I

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

Seconds to time interval.

Converts from seconds to system ticks number.

Note

The result is rounded upward to the next tick boundary.

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

Parameters

| in | secs | number of seconds |
Returns
The number of ticks.

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

6.21.2.6 TIME_MS2I

#define TIME_MS2I(
msecs )

Value:
((sysinterval_t)((((time_conv_t)(msecs) * (time_conv_t)CH_CFG_ST_FREQUENCY) + 999) / (time_conv_t)1000))

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

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

Parameters

| in | msecs | number of milliseconds |

Returns
The number of ticks.

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

6.21.2.7 TIME_US2I

#define TIME_US2I(
usecs )

Value:
((sysinterval_t)((((time_conv_t)(usecs) * (time_conv_t)CH_CFG_ST_FREQUENCY) + 999999) / (time_conv_t)1000000))

Microseconds to time interval.
Converts from microseconds to system ticks number.
Note

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

Parameters

```
in  usecs  number of microseconds
```

Returns

The number of ticks.

Function Class:

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

6.21.2.8 TIME_I2S

```c
#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.
#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.

#define TIME_I2US(
  interval )

Value:

```
[time_msecs_t]\{((time_conv_t)(interval) * (time_conv_t)1000000) + \ 
  (time_conv_t)CH_CFG_ST_FREQUENCY - (time_conv_t)1) / \ 
  (time_conv_t)CH_CFG_ST_FREQUENCY\}
```

Time interval to microseconds.

Converts from system ticks number to microseconds.

Note

The result is rounded up to the next microsecond boundary.

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

in  interval  interval in ticks

Returns

The number of microseconds.

Function Class:

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

6.21.3  Typedef Documentation

6.21.3.1  systime_t

typedef uint64_t systime_t

Type of system time.

Note

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

6.21.3.2  sysinterval_t

typedef uint64_t sysinterval_t

Type of time interval.

Note

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

6.21.3.3  systimestamp_t

typedef uint64_t systimestamp_t

Type of a time stamp.
6.21.3.4  time_secs_t

typedef uint32_t time_secs_t
Type of seconds.

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

6.21.3.5  time_msecs_t

typedef uint32_t time_msecs_t
Type of milliseconds.

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

6.21.3.6  time_usecs_t

typedef uint32_t time_usecs_t
Type of microseconds.

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

6.21.3.7  time_conv_t

typedef uint64_t time_conv_t
Type of time conversion variable.

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

6.21.4  Function Documentation

6.21.4.1  chTimeS2I()

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

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

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

| in | secs | number of seconds |

Returns

The number of ticks.

Function Class:

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

### 6.21.4.2 chTimeMS2I()

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

Milliseconds to time interval.

Converts from milliseconds to system ticks number.

Note

The result is rounded upward to the next tick boundary.

Parameters

| in | msec | number of milliseconds |

Returns

The number of ticks.

Function Class:

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

### 6.21.4.3 chTimeUS2I()

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

Microseconds to time interval.

Converts from microseconds to system ticks number.

Note

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

| in  | usec | number of microseconds |

Returns

The number of ticks.

Function Class:

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

6.21.4.4 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.

6.21.4.5 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.
Parameters

<table>
<thead>
<tr>
<th></th>
<th>interval</th>
<th>interval in ticks</th>
</tr>
</thead>
</table>

Returns

The number of milliseconds.

Function Class:

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

### 6.21.4.6 chTimeI2US()

```c
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

<table>
<thead>
<tr>
<th></th>
<th>interval</th>
<th>interval in ticks</th>
</tr>
</thead>
</table>

Returns

The number of microseconds.

Function Class:

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

### 6.21.4.7 chTimeAddX()

```c
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.

### 6.21.4.8 chTimeDiffX()

```c
static sysinterval_t chTimeDiffX ( systick_t start, systick_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.

### 6.21.4.9 chTimesInRangeX()

```c
static bool chTimesInRangeX ( systick_t time, systick_t start, systick_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>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>time</strong></td>
<td>the time to be verified</td>
</tr>
<tr>
<td><strong>start</strong></td>
<td>the start of the time window (inclusive)</td>
</tr>
<tr>
<td><strong>end</strong></td>
<td>the end of the time window (non inclusive)</td>
</tr>
</tbody>
</table>

Return values

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

Function Class:

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

6.21.4.10  \texttt{chTimeStampAddX()} \newline

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

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

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>stamp</strong></td>
<td>base time stamp</td>
</tr>
<tr>
<td><strong>interval</strong></td>
<td>interval to be added</td>
</tr>
</tbody>
</table>

Returns

The new time stamp.

Function Class:

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

6.21.4.11  \texttt{chTimeStampDiffX()} \newline

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

Subtracts two time stamps returning an interval.

Note

Intervals can then be used for converting in absolute time.
6.21 Time and Intervals

Parameters

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

Returns

The interval representing the time stamps difference.

Function Class:

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

6.21.4.12 chTimeStampIsInRangeX()

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

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

Note

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

Parameters

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

Return values

true if the current time stamp is within the specified time stamp window.
false if the current time stamp is not within the specified time stamp window.

Function Class:

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

6.22.1 Detailed Description

Time and Virtual Timers related APIs and services.

Functions

- 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 virtual timer in a virtual timers list.
- void chVTDoSetI (virtual_timer_t ∗vtp, sysinterval_t delay, vtfunc_t vtfunc, void ∗par)
  Enables a one-shot virtual timer.
- void chVTDoSetContinuousI (virtual_timer_t ∗vtp, sysinterval_t delay, vtfunc_t vtfunc, void ∗par)
  Enables a continuous virtual timer.
- void chVTDoResetI (virtual_timer_t ∗vtp)
  Disables a Virtual Timer.
- systime_t chVTGetRemainingIntervalI (virtual_timer_t ∗vtp)
  Returns the remaining time interval before next timer trigger.
- void chVTDoTickI (void)
  Virtual timers ticker.
- systimestamp_t chVTGetTimeStampI (void)
  Generates a monotonic time stamp.
- void chVTResetTimeStampI (void)
  Resets and re-synchronizes the time stamps monotonic counter.
- static void chVTObjectInit (virtual_timer_t ∗vtp)
  Initializes a virtual_timer_t object.
- static systime_t chVTGetSystemTimeX (void)
  Current system time.
- static systime_t chVTGetSystemTime (void)
  Current system time.
- static systime_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 chVTSet (virtual_timer_t *vtp, sysinterval_t delay, vtfunc_t vtfunc, void *par)
  Enables a one-shot virtual timer.
• static void chVTSet (virtual_timer_t *vtp, sysinterval_t delay, vtfunc_t vtfunc, void *par)
  Enables a one-shot virtual timer.
• static void chVTSetContinuousI (virtual_timer_t *vtp, sysinterval_t delay, vtfunc_t vtfunc, void *par)
  Enables a continuous virtual timer.
• static void chVTSetContinuous (virtual_timer_t *vtp, sysinterval_t delay, vtfunc_t vtfunc, void *par)
  Enables a continuous virtual timer.
• static sysinterval_t chVTGetReloadIntervalX (virtual_timer_t *vtp)
  Returns the current reload value.
• static void chVTSetReloadIntervalX (virtual_timer_t *vtp, sysinterval_t reload)
  Changes a timer reload time interval.
• static systimestamp_t chVTGetTimeStamp (void)
  Generates a monotonic time stamp.
• static void chVTResetTimeStamp (void)
  Resets and re-synchronizes the time stamps monotonic counter.
• static void __vt_object_init (virtual_timers_list_t *vtlp)
  Virtual Timers instance initialization.

6.22.2 Function Documentation

6.22.2.1 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

<table>
<thead>
<tr>
<th>in</th>
<th>now</th>
<th>last known system time</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>delay</td>
<td>delay over now</td>
</tr>
</tbody>
</table>
Here is the call graph for this function:

![Call Graph Image]

### 6.22.2.2 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.
6.22 Virtual Timers

Note

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

Here is the call graph for this function:

6.22.2.3 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:

6.22.2.4 chVTDoSetI()

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

Enables a one-shot virtual timer.

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

Precondition

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

Note

The callback function is invoked from interrupt context.

Parameters

| out | vtp | the virtual_timer_t structure pointer |

ChibiOS/RT
Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>delay</th>
<th>the number of ticks before the operation timeouts, the special values are handled as follow:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>• <strong>TIME_INFINITE</strong> is allowed but interpreted as a normal time specification.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <strong>TIME_IMMEDIATE</strong> this value is not allowed.</td>
</tr>
</tbody>
</table>

| in | vtfunc | the timer callback function. After invoking the callback the timer is disabled and the structure can be disposed or reused. |

| in | par | a parameter that will be passed to the callback function |

Function Class:

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

Here is the call graph for this function:

![Call Graph](image_url)

6.22.2.5 **chVTDoSetContinuousI()**

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

Enables a continuous virtual timer.

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

Precondition

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

Note

The callback function is invoked from interrupt context.
Parameters

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

Function Class:

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

Here is the call graph for this function:

```
6.22.2.6  chVTDoResetI()

void chVTDoResetI (  
  virtual_timer_t * vtp )

Disables a Virtual Timer.

Precondition

The timer must be in armed state before calling this function.
```
6.22 Virtual Timers

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:

```
chVTDoResetI
  ch_dlist_dequeue
  ch_dlist_isempty
  ch_dlist_isfirst
  ch_dlist_remove_first
  chDbgCheckClassI
  chTimeDiffX
  chVTGetSystemTimeX
  chVTIsArmedI
  vt_set_alarm
  chSysHalt
  chRFCUCollectFaultsI
  chTimeAddX
  chVTGetRemainingIntervalI
```

6.22.2.7 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.
6.22.2.8 chVTDoTick()

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:

6.22.2.9 *chVTGetTimeStampI()*

```c
systimestamp_t chVTGetTimeStampI ( void )
```

Generates a monotonic time stamp.

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

**Note**

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

The time stamp.

Function Class:

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

Here is the call graph for this function:

6.22.2.10  chVTResetTimeStampI()

```c
void chVTResetTimeStampI ( void )
```

Resets and re-synchronizes the time stamps monotonic counter.

Function Class:

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

Here is the call graph for this function:
6.22 Virtual Timers

6.22.2.11 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.

6.22.2.12 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.
6.22.2.13 chVTGetSystemTime()

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

Current system time.

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

**Note**

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

**Returns**

The system time in ticks.

**Function Class:**

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

Here is the call graph for this function:

```
chVTGetSystemTime
chSysLock
chSysUnlock
chVTGetSystemTimeX
__dbg_check_lock
__stats_start_measure
_crit_thd
__dbg_check_unlock
__stats_stop_measure
_crit_thd
```

6.22.2.14 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:

![Call Graph]

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

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>current time within the specified time window.</td>
</tr>
<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.
Here is the call graph for this function:

```
chVTIsSystemTimeWithinX
   └── chTimeIsInRangeX
      └── chVTGetSystemTimeX
```

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

**Return values**

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

### 6.22.2.17 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_TIMEDELTA` ticks.

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

**Parameters**

| 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-url)

### 6.22.2.18 chVTIsArmed()  

```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 `chVTOBJECTInit()` or `chVTDOSetI()`.

**Parameters**

<table>
<thead>
<tr>
<th></th>
<th>vtp</th>
<th>the virtual_timer_t structure pointer</th>
</tr>
</thead>
</table>

**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
```

### 6.22.2.19 chVTIsArmed()

```c
cstatic bool chVTIsArmed (const virtual_timer_t *vtp) __attribute__((inline), __attribute__((static)));
```

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

Precondition

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

Parameters

| `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
      __dbg_check_lock
      __stats_start_measure
      __stats_stop_measure
    chVTIsArmedI
  chVTIsArmedI
    chSysUnlock
      __dbg_check_unlock
      __stats_start_measure
      __stats_stop_measure
    chVTIsArmedI
  chDbgCheckClassI
```

6.22.2.20 chVTResetI()

```c
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

- **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:

6.22.2.21  chVTReset()

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

Disables a Virtual Timer.

Note

The timer is first checked and disabled only if armed.

Precondition

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

Parameters

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

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

Here is the call graph for this function:

```
chVTReset
chSysLock
chSysUnlock
chVTResetI
__dbg_check_lock
__stats_start_measure
_crit_thd
__dbg_check_unlock
__stats_stop_measure
_crit_thd
chVTDoResetI
chVTIsArmedI
```

### 6.22.2.22 chVTSetI()

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

Enables a one-shot virtual timer.

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

**Precondition**

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

**Parameters**

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

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

Here is the call graph for this function:

![Call Graph](image)

### 6.22.2.23 chVTSet()

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

Enables a one-shot virtual timer.

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

**Precondition**

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

**Parameters**

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

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

Here is the call graph for this function:

6.22.2.24 chVTSetContinuousI()

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

Enables a continuous virtual timer.

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

Precondition

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

Parameters

| in | vtp | the virtual_timer_t structure pointer |
6.22 Virtual Timers

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>delay</th>
<th>the number of ticks before the operation timeouts, the special values are handled as follow:</th>
</tr>
</thead>
<tbody>
<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>vfunc</td>
<td>the timer callback function. After invoking the callback the timer is disabled and the structure</td>
</tr>
<tr>
<td></td>
<td></td>
<td>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:

```
chVTSetContinuous()  chVTDoSetContinuous()  chVTReset()  chDbgCheckClass()  vt_enqueue  chVTDoReset()  chVTIsArmed()  chVTObjectInit()
```

6.22.2.25 chVTSetContinuous()

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

Enables a continuous virtual timer.

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

Precondition

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

Parameters

<table>
<thead>
<tr>
<th>in</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>vfunc</td>
<td>the timer callback function. After invoking the callback the timer is disabled and the structure</td>
</tr>
<tr>
<td></td>
<td></td>
<td>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:

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

Here is the call graph for this function:

```
chVTSetContinuous
chSysLock
__dbg_check_lock
__stats_start_measure
__stats_stop_measure
_chthd
__dbg_check_unlock
chSysUnlock
chVTDoSetContinuous
chVTResetI

6.22.2.26 chVTGetReloadIntervalX()

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

Returns the current reload value.

Parameters

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

Returns

The reload value.

Function Class:

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

6.22.27 chVTSetReloadIntervalX()

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

Changes a timer reload time interval.

Note

This function is meant to be called from a timer callback, it does nothing in any other context. Calling this function from a one-shot timer callback turns it into a continuous timer.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>vtp</th>
<th>the virtual_timer_t structure pointer</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>reload</td>
<td>the new reload value, zero means no reload</td>
</tr>
</tbody>
</table>

Function Class:

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

6.22.28 chVTGetTimeStamp()

static systimestamp_t chVTGetTimeStamp ( 
    void ) [inline], [static]

Generates a monotonic time stamp.

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

Note

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

Returns

The time stamp.
Function Class:

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

Here is the call graph for this function:

6.22.2.29 chVTResetTimeStamp()

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

Resets and re-synchronizes the time stamps monotonic counter.
Function Class:

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

Here is the call graph for this function:

```
chVTResetTimeStamp
chSysLock
chSysUnlock
chVTResetTimeStampI
__dbg_check_lock
__stats_start_measure
_crit_thd
__dbg_check_unlock
__stats_stop_measure
_crit_thd
chDbgCheckClassI
chVTGetSystemTimeX
```

### 6.22.2.30 __vt_object_init()

```c
static void __vt_object_init ( virtual_timers_list_t * vtlp ) [inline], [static]
```

Virtual Timers instance initialization.

Note

Internal use only.

Parameters

| out | vtlp | pointer to the virtual_timers_list_t structure |
Function Class:

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

Here is the call graph for this function:
6.23 Threads

6.23.1 Detailed Description

Threads related APIs and services.

Operation mode

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

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

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

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

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

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

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

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

Threads queues

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

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

Working Areas

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

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

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

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

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

Threads initializers

- `#define THD_DESCRIPTOR(name, wbase, wend, prio, funcp, arg)`
  
  Thread descriptor initializer with no affinity.

- `#define THD_DESCRIPTOR_AFFINITY(name, wbase, wend, prio, funcp, arg, oip)`
  
  Thread descriptor initializer with no affinity.

Macro Functions

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

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

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

Typedefs

- `typedef void(* tfunc_t)(void *p)`
  
  Thread function.

Data Structures

- `struct thread_descriptor_t`
  
  Type of a thread descriptor.

Functions

- `thread_t *__thd_object_init (os_instance_t *oip, thread_t *tp, const char *name, tprio_t prio)`
  
  Initializes a thread structure.

- `void __thd_stackfill (uint8_t *startp, uint8_t *endp)`
  
  Stack fill utility.

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

- `thread_t *chThdCreateSuspendedI (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.
6.23 Threads

- 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)
  
  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.

- 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.

### 6.23.2 Macro Definition Documentation

#### 6.23.2.1 __THREADS_QUEUE_DATA

```c
#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

#### 6.23.2.2 THREADS_QUEUE_DECL

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

Static threads queue object initializer.

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

**Parameters**

- **in** name the name of the threads queue variable
6.23.2.3 THD_WORKING_AREA_SIZE

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

Calculates the total Working Area size.

Parameters

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

Returns

The total used memory in bytes.

Function Class:

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

6.23.2.4 THD_WORKING_AREA

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

Static working area allocation.

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

Parameters

| in | s | the name to be assigned to the stack array |
| in | n | the stack size to be assigned to the thread |

Function Class:

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

6.23.2.5 THD_WORKING_AREA_BASE

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

Base of a working area casted to the correct type.
6.23.2.6 THD_WORKING_AREA_END

```c
#define THD_WORKING_AREA_END(s) THD_WORKING_AREA_BASE(s) + (sizeof (s) / sizeof (stkalign_t))
```

End of a working area casted to the correct type.

Parameters

| in  | s     | name of the working area |

6.23.2.7 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.

6.23.2.8 THD_DESCRIPTOR

```c
#define THD_DESCRIPTOR(name, wbase, wend, prio, funcp, arg)
```

Value:

```c
{name}, \[name],
```

Parameters

| in  | s     | name of the working area |
Thread descriptor initializer with no affinity.
6.23.2.9 THD_DESCRIPTOR_AFFINITY

```c
#define THD_DESCRIPTOR_AFFINITY(
    name,
    wbase,
    wend,
    prio,
    funcp,
    arg,
    oip )
```

Value:

```c
{ (name),
  (wbase),
  (wend),
  (prio),
  (funcp),
  (arg),
  (oip) }
```

Thread descriptor initializer with no affinity.

6.23.2.10 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.

6.23.2.11 chThdSleepMilliseconds

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

Delays the invoking thread for the specified number of milliseconds.

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

Parameters

| in  | 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.

6.23.2.12 chThdSleepMicroseconds

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

Delays the invoking thread for the specified number of microseconds.
Note

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

Parameters

| in  | 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.

6.23.3 Typedef Documentation

6.23.3.1 tfunc_t

typedef void(* tfunc_t) (void *)

Thread function.

6.23.4 Function Documentation

6.23.4.1 __thd_object_init()

thread_t *__thd_object_init (  
    os_instance_t * oip,  
    thread_t * tp,  
    const char * name,  
    tprio_t prio )

Initializes a thread structure.

Note

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

Parameters

| in  | oip | pointer to the OS instance |
| in  | tp  | pointer to the thread |
| in  | name | thread name |
| in  | prio | the priority level for the new thread |
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      |
| _thd_object_init  |   ch_queue_init   |
|                   |   chTMObjectInit  |
+-------------------+
```

6.23.4.2  __thd_stackfill()

```c
void __thd_stackfill (  
    uint8_t * startp,  
    uint8_t * endp )
```

Stack 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>
</tbody>
</table>

Function Class:

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

6.23.4.3  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 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:
6.23.4.4 chThdCreateSuspended()

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

Creates a new thread into a static memory area.

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

Postcondition

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

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

Note

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

Parameters

- **out** `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:
6.23.4.5 chThdCreateI()

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

Creates a new thread into a static memory area.

The new thread is initialized and make ready to execute.

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:
6.23.4.6 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

- `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:

6.23.4.7 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

| out  | wsp  | pointer to a working area dedicated to the thread stack |
| in   | size | size of the working area |
| 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.

Function Class:

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

Here is the call graph for this function:

6.23.4.8 chThdStart()

thread_t * chThdStart ( thread_t * tp )

Resumes a thread created with chThdCreateI().
Parameters

\texttt{tp} pointer to the thread

Returns

The pointer to the \texttt{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:

6.23.4.9 \texttt{chThdAddRef()}

\verb|thread_t * chThdAddRef ( thread_t * tp )|

 Adds a reference to a thread object.

Precondition

The configuration option \texttt{CH_CFG_USE_REGISTRY} must be enabled in order to use this function.
Parameters

- **in tp** pointer to the thread

Returns

The same thread pointer passed as parameter representing the new reference.

Function Class:

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

Here is the call graph for this function:

```
<table>
<thead>
<tr>
<th>chThdAddRef</th>
<th>__dbg_check_lock</th>
</tr>
</thead>
<tbody>
<tr>
<td>chSysLock</td>
<td>__stats_start_measure</td>
</tr>
<tr>
<td></td>
<td>_crit_thd</td>
</tr>
<tr>
<td>chSysUnlock</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>
```

### 6.23.4.10 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:

6.23.4.11 chThdExit()

void chThdExit ( msg_t msg )

Terminates the current thread.

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

Postcondition

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

Parameters

in 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:

6.23.4.12 chThdExitS()

void chThdExitS (  
    msg_t  msg  )

Terminates the current thread.

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

Postcondition

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

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

<table>
<thead>
<tr>
<th></th>
<th>msg</th>
<th>thread exit code</th>
</tr>
</thead>
</table>

Function Class:

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

Here is the call graph for this function:

![Call Graph](image)

**6.23.4.13 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:

6.23.4.14 chThdSetPriority()

t prio_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:

6.23.4.15 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

| 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:

![Call Graph](image)

6.23.4.16 chThdSleep()

```c
void chThdSleep ( sysinterval_t time )
```

Suspends the invoking thread for the specified time.

Parameters

| In | time | the delay in system ticks, the special values are handled as follow:
|----|------|------------------------------------------------------------------
|    |      | • `TIME_INFINITE` the thread enters an infinite sleep state.       |
|    |      | • `TIME_IMMEDIATE` this value is not allowed.                      |

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:

6.23.4.17 chThdSleepUntil()

```
void chThdSleepUntil (  
    systime_t  time  )
```

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

Note
The function has no concept of "past", all specifiable times are in the future, this means that if you call this function exceeding your calculated intervals then the function will return in a far future time, not immediately.

See also

chThdSleepUntilWindowed()

Parameters

| in | time | absolute system time |
Function Class:

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

Here is the call graph for this function:

```
chThdSleepUntil
chSysLock
chSysUnlock
chThdSleepS
chTimeDiffX
chVTGetSystemTimeX
__dbg_check_lock
__stats_start_measure
_crit_thd
__dbg_check_unlock
__stats_stop_measure
_crit_thd
chSchGoSleepTimeoutS
```

6.23.4.18 chThdSleepUntilWindowed()

```
systime_t chThdSleepUntilWindowed ( systime_t prev,
       systime_t next )
```

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

Note

The system time is assumed to be between `prev` and `next` else the call is assumed to have been called outside the allowed time interval, in this case no sleep is performed.

See also

- `chThdSleepUntil()`

Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td><code>prev</code></td>
<td>absolute system time of the previous deadline</td>
</tr>
<tr>
<td>in</td>
<td><code>next</code></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:

6.23.4.19 chThdYield()

```c
void chThdYield ( )
```

Yields the time slot.

Yields the CPU control to the next thread in the ready list with equal priority, if any.
Function Class:

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

Here is the call graph for this function:

6.23.4.20 chThdSuspendS()

```c
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

- **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:

![Call Graph]

6.23.4.21 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>in</th>
<th><code>trp</code></th>
<th>a pointer to a thread reference object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td><code>timeout</code></td>
<td>the timeout in system ticks, the special values are handled as follow:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <code>TIME_INFINITE</code> the thread enters an infinite sleep state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <code>TIME_IMMEDIATE</code> the thread is not suspended and the function returns <code>MSG_TIMEOUT</code> as if a timeout occurred.</td>
</tr>
</tbody>
</table>

**Returns**

The wake up message.

**Return values**

`MSG_TIMEOUT` if the operation timed out.
6.23 Threads

Function Class:

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

Here is the call graph for this function:

![Call Graph](image)

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

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

Here is the call graph for this function:

![Call Graph](image)

6.23.4.23  chThdResumeS()

```c
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>
6.23 Threads

Function Class:

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

Here is the call graph for this function:

![Call Graph Diagram]

6.23.4.24 chThdResume()

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

Wakes up a thread waiting on a thread reference object.

Note

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

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>trp</td>
<td>a pointer to a thread reference object</td>
</tr>
<tr>
<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 Image]

6.23.4.25 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

<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>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 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)

### 6.23.4.26 chThdDequeueNextI()

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

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

**Parameters**

| in | tqp | pointer to the threads queue object |
| in | msg | the message code |
Function Class:

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

Here is the call graph for this function:

![Call Graph](image)

### 6.23.4.27  `chThdDequeueAllI()`

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

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

**Parameters**

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

Function Class:

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

Here is the call graph for this function:

![Call Graph](image)
6.23.4.28 chThdGetSelfX()

static thread_t* chThdGetSelfX ()
{ void ) [inline], [static]
Returns a pointer to the current thread_t.

Returns
A pointer to the current thread.

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

6.23.4.29 chThdGetPriorityX()

static tprio_t chThdGetPriorityX ()
{ void ) [inline], [static]
Returns the current thread priority.

Note
Can be invoked in any context.

Returns
The current thread priority.

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

Here is the call graph for this function:

```
chThdGetPriorityX
  | ^
  |     chThdGetSelfX
```

6.23.4.30 chThdGetTicksX()

static systime_t chThdGetTicksX ()
{ thread_t * tp ) [inline], [static]
Returns the number of ticks consumed by the specified thread.

Note
This function is only available when the CH_DBG_THREADS_PROFILING configuration option is enabled.
Parameters

| in  | tp  | pointer to the thread |

Returns

The number of consumed system ticks.

Function Class:

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

6.23.4.31  chThdGetWorkingAreaX()

```c
static stkalign_t* chThdGetWorkingAreaX ( thread_t * tp ) [inline], [static]
```

Returns the working area base of the specified thread.

Parameters

| in  | tp  | pointer to the thread |

Returns

The working area base pointer.

Function Class:

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

6.23.4.32  chThdTerminatedX()

```c
static bool chThdTerminatedX ( thread_t * tp ) [inline], [static]
```

Verifies if the specified thread is in the CH_STATE_FINAL state.

Parameters

| in  | tp  | pointer to the thread |
6.23 Threads

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.

6.23.4.33 chThdShouldTerminateX()

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:

![Call graph for chThdShouldTerminateX](image)

6.23.4.34 chThdStartI()

static thread_t* chThdStartI ( thread_t * tp ) [inline], [static]

Resumes a thread created with chThdCreateI().
216 Module Documentation

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>tp</th>
<th>pointer to the thread</th>
</tr>
</thead>
</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:

```
chThdStartI  chSchReadyI  __sch_ready_behind  chDbgCheckClassI  chSysNotifyInstance
```

6.23.4.35 chThdSleepS()

```c
static void chThdSleepS (sysinterval_t ticks)
```

Suspends the invoking thread for the specified number of ticks.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>ticks</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>

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](image)

### 6.23.4.36 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](image)

### 6.23.4.37 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**

| false | if the queue is not empty. |
| true  | if the queue is empty. |

**Function Class:**

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

Here is the call graph for this function:

![Call Graph]

---

**6.23.4.38  chThdDoDequeueNextI()**

```c
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:
6.24 Time Measurement

6.24.1 Detailed Description

Time Measurement APIs and services.

Macros

- \#define TM_CALIBRATION_LOOP 4U
  
  Number of iterations in the calibration loop.

Data Structures

- struct tm_calibration_t
  
  Type of a time measurement calibration data.

- struct time_measurement_t
  
  Type of a Time Measurement object.

Functions

- void chTMObjectInit (time_measurement_t *tmp)
  
  Initializes a TimeMeasurement object.

- NOINLINE void chTMStartMeasurementX (time_measurement_t *tmp)
  
  Starts a measurement.

- NOINLINE void chTMStopMeasurementX (time_measurement_t *tmp)
  
  Stops a measurement.

- NOINLINE void chTMChainMeasurementToX (time_measurement_t *tmp1, time_measurement_t *tmp2)
  
  Stops a measurement and chains to the next one using the same time stamp.

- static void __tm_calibration_object_init (tm_calibration_t *tcp)
  
  Time measurement initialization.

6.24.2 Macro Definition Documentation

6.24.2.1 TM_CALIBRATION_LOOP

#define TM_CALIBRATION_LOOP 4U

Number of iterations in the calibration loop.

Note

This is required in order to assess the best result in architectures with instruction cache.

6.24.3 Function Documentation

6.24.3.1 chTMObjectInit()

void chTMObjectInit ( 
    time_measurement_t * tmp )

Initializes a TimeMeasurement object.
6.24 Time Measurement

Parameters

<table>
<thead>
<tr>
<th>type</th>
<th>name</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>tmp</td>
<td>pointer to a TimeMeasurement structure</td>
</tr>
</tbody>
</table>

Function Class:

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

6.24.3.2 chTMStartMeasurementX()

NOINLINE void chTMStartMeasurementX ( 
  
  time_measurement_t _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>in,out</td>
<td>tmp</td>
<td>pointer to a TimeMeasurement structure</td>
</tr>
</tbody>
</table>

Function Class:

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

6.24.3.3 chTMStopMeasurementX()

NOINLINE void chTMStopMeasurementX ( 
  
  time_measurement_t _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>in,out</td>
<td>tmp</td>
<td>pointer to a time_measurement_t structure</td>
</tr>
</tbody>
</table>
6.24.3.4  chTMChainMeasurementToX()

NOINLINE void chTMChainMeasurementToX (  
    time_measurement_t * tmp1,  
    time_measurement_t * tmp2  )

Stops a measurement and chains to the next one using the same time stamp.

Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in,out</td>
<td>tmp1 pointer to the time_measurement_t structure to be stopped</td>
</tr>
<tr>
<td>in,out</td>
<td>tmp2 pointer to the time_measurement_t structure to be started</td>
</tr>
</tbody>
</table>

Function Class:

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

6.24.3.5  __tm_calibration_object_init()

static void __tm_calibration_object_init (  
    tm_calibration_t * tcp  ) [inline], [static]

Time measurement initialization.

Note

Internal use only.

Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>tcp pointer to the tm_calibration_t structure</td>
</tr>
</tbody>
</table>
Function Class:

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

Here is the call graph for this function:
6.25 Synchronization

6.25.1 Detailed Description

Synchronization services.

Modules

- Counting Semaphores
- Mutexes
- Condition Variables
- Event Flags
- Synchronous Messages
6.26 Counting Semaphores

6.26.1 Detailed Description

Semaphores related APIs and services.

Operation mode

Semaphores are a flexible synchronization primitive, ChibiOS/RT implements semaphores in their "counting semaphores" variant as defined by Edsger Dijkstra plus several enhancements like:

- Wait operation with timeout.
- Reset operation.
- Atomic wait+signal operation.
- Return message from the wait operation (OK, RESET, TIMEOUT).

The binary semaphores variant can be easily implemented using counting semaphores. Operations defined for semaphores:

- **Signal**: The semaphore counter is increased and if the result is non-positive then a waiting thread is removed from the semaphore queue and made ready for execution.
- **Wait**: The semaphore counter is decreased and if the result becomes negative the thread is queued in the semaphore and suspended.
- **Reset**: The semaphore counter is reset to a non-negative value and all the threads in the queue are released.

Semaphores can be used as guards for mutual exclusion zones (note that mutexes are recommended for this kind of use) but also have other uses, queues guards and counters for example. Semaphores usually use a FIFO queuing strategy but it is possible to make them order threads by priority by enabling `CH_CFG_USE_SEMAPHORES_PRIORITY` in `chconf.h`.

Precondition

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

Macros

- `#define __SEMAPHORE_DATA(name, n) {__CH_QUEUE_DATA(name.queue), n}`
  
  Data part of a static semaphore initializer.

- `#define SEMAPHORE_DECL(name, n) semaphore_t name = __SEMAPHORE_DATA(name, n)`
  
  Static semaphore initializer.

Typedefs

- `typedef struct ch_semaphore semaphore_t`

  Semaphore structure.
Data Structures

- struct ch_semaphore
  Semaphore structure.

Functions

- void chSemObjectInit (semaphore_t *sp, cnt_t n)
  Initializes a semaphore with the specified counter value.
- void chSemResetWithMessage (semaphore_t *sp, cnt_t n, msg_t msg)
  Performs a reset operation on the semaphore.
- void chSemResetWithMessageI (semaphore_t *sp, cnt_t n, msg_t msg)
  Performs a reset operation on the semaphore.
- msg_t chSemWait (semaphore_t *sp)
  Performs a wait operation on a semaphore.
- msg_t chSemWaitS (semaphore_t *sp)
  Performs a wait operation on a semaphore.
- msg_t chSemWaitTimeout (semaphore_t *sp, sysinterval_t timeout)
  Performs a wait operation on a semaphore with timeout specification.
- msg_t chSemWaitTimeoutS (semaphore_t *sp, sysinterval_t timeout)
  Performs a wait operation on a semaphore with timeout specification.
- void chSemSignal (semaphore_t *sp)
  Performs a signal operation on a semaphore.
- void 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 *sw)
  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.

6.26.2 Macro Definition Documentation

6.26.2.1 __SEMAPHORE_DATA

#define __SEMAPHORE_DATA(
  name,
  n)
  (__CH_QUEUE_DATA(name.queue), n)

Data part of a static semaphore initializer.

This macro should be used when statically initializing a semaphore that is part of a bigger structure.
### 6.26.2.2 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</th>
<th>name</th>
<th>the name of the semaphore variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>n</td>
<td>the counter initial value, this value must be non-negative</td>
</tr>
</tbody>
</table>

### 6.26.3 Typedef Documentation

#### 6.26.3.1 semaphore_t

```c
typedef struct ch_semaphore semaphore_t
```

Semaphore structure.

### 6.26.4 Function Documentation

#### 6.26.4.1 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  
```

#### 6.26.4.2 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>

---
Function Class:

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

Here is the call graph for this function:

![Call Graph]

6.26.4.3 chSemResetWithMessageI()

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

Performs a reset operation on the semaphore.

Postcondition

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

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

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

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

Here is the call graph for this function:

---

6.26.4.4  **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>
<tbody>
<tr>
<td>MSG_RESET</td>
<td>if the semaphore has been reset using <code>chSemReset()</code>.</td>
</tr>
</tbody>
</table>
6.26 Counting Semaphores

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

Here is the call graph for this function:

![Call Graph]

6.26.4.5 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>
</table>
| MSG_RESET | if the semaphore has been reset using `chSemReset()`.

Function Class:

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

Here is the call graph for this function:

![Call Graph](image)

### 6.26.4.6 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>• <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

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:

```
chSemWaitTimeout
  |             |
  v             v
chSemWaitTimeoutS  chDbgCheckClassS
  |             |                 |
  v             v                 v
ch_queue_isempty  ch_thd_get_selfX
  |             |             |
  v             v             v
__dbg_check_lock  __stats_start_measure
  |             |             |
  v             v             v
_crit_thd

6.26.4.7 chSemWaitTimeoutS()

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

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

<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>
<tr>
<td>MSG_TIMEOUT</td>
<td>if the semaphore has not been signaled or reset 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:

```
chSemWaitTimeoutS
  |________________________|
  |                      |
  |________________________|
  |                      |
  v                      v
ch_queue_isempty ch_queue_notempty
  |________________________|
  |                      |
  |________________________|
  |                      |
  v                      v
chDbgCheckClassS chThdGetSelfX
  |________________________|
  |                      |
  |________________________|
  |                      |
  v                      v
chSysHalt
```

### 6.26.4.8 chSemSignal()

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

Performs a signal operation on a semaphore.
6.26 Counting Semaphores

Parameters

| in | sp | pointer to a semaphore_t structure |

Function Class:

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

Here is the call graph for this function:

6.26.4.9 chSemSignalI()

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

Performs a signal operation on a semaphore.

Postcondition

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

| in  | sp  | pointer to a semaphore_t structure |

Function Class:

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

Here is the call graph for this function:

6.26.4.10  chSemAddCounterI()

```
void chSemAddCounterI ( 
    semaphore_t * sp, 
    cnt_t n )
```

Adds the specified value to the semaphore counter.

Postcondition

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

Parameters

| in  | sp  | pointer to a semaphore_t structure |
| in  | n   | value to be added to the semaphore counter. The value must be positive. |
Function Class:

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

Here is the call graph for this function:

---

**6.26.4.11 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 wait on</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 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:
Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

![call graph image]

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

```
chSemReset
  chSemResetWithMessage
  chSchRescheduleS
  chSemResetWithMessageI
  chSysLock
  chSysUnlock
```

### 6.26.4.13 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> pointer to a <code>semaphore_t</code> structure</td>
</tr>
<tr>
<td><code>in</code></td>
<td><code>n</code> the new value of the semaphore counter. The value must be non-negative.</td>
</tr>
</tbody>
</table>
Function Class:

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

Here is the call graph for this function:

```
chSemResetI chSemResetWithMessageI
ch_queue_isempty
ch_queue_lifo_remove
ch_queue_notempty
chDbgCheckClassI
chSchReadyI
```

### 6.26.4.14 chSemFastWaitI()

```c
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**

<table>
<thead>
<tr>
<th><strong>in</strong></th>
<th><strong>sp</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>pointer to a semaphore_t structure</strong></td>
<td></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:

```
chSemFastWaitI chDbgCheckClassI chSysHalt
```
6.26.4.15  

**chSemFastSignalI()**

```c
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:

```
chSemFastSignalI  chDbgCheckClassI  chSysHalt
```

6.26.4.16  

**chSemGetCounterI()**

```c
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:
6.27 Mutexes

6.27.1 Detailed Description

Mutexes related APIs and services.

Operation mode

A mutex is a threads synchronization object that can be in two distinct states:

- Not owned (unlocked).
- Owned by a thread (locked).

Operations defined for mutexes:

- **Lock**: The mutex is checked, if the mutex is not owned by some other thread then it is associated to the locking thread else the thread is queued on the mutex in a list ordered by priority.
- **Unlock**: The mutex is released by the owner and the highest priority thread waiting in the queue, if any, is resumed and made owner of the mutex.

Constraints

In ChibiOS/RT the Unlock operations must always be performed in lock-reverse order. This restriction both improves the performance and is required for an efficient implementation of the priority inheritance mechanism. Operating under this restriction also ensures that deadlocks are no possible.

Recursive mode

By default mutexes are not recursive, this mean that it is not possible to take a mutex already owned by the same thread. It is possible to enable the recursive behavior by enabling the option `CH_CFG_USE_MUTEXES_RECURSIVE`.

The priority inversion problem

The mutexes in ChibiOS/RT implements the full priority inheritance mechanism in order handle the priority inversion problem. When a thread is queued on a mutex, any thread, directly or indirectly, holding the mutex gains the same priority of the waiting thread (if their priority was not already equal or higher). The mechanism works with any number of nested mutexes and any number of involved threads. The algorithm complexity (worst case) is $N$ with $N$ equal to the number of nested mutexes.

Precondition

In order to use the mutex APIs the `CH_CFG_USE_MUTEXES` option must be enabled in `chconf.h`.

Postcondition

Enabling mutexes requires 5-12 (depending on the architecture) extra bytes in the `thread_t` structure.
Macros

- `#define __MUTEX_DATA(name) {__CH_QUEUE_DATA(name.queue), NULL, NULL, 0}`
  Data part of a static mutex initializer.
- `#define MUTEX_DECL(name) mutex_t name = __MUTEX_DATA(name)`
  Static mutex initializer.

Typedefs

- `typedef struct ch_mutex mutex_t`
  Type of a mutex structure.

Data Structures

- `struct ch_mutex`
  Mutex structure.

Functions

- `void chMtxObjectInit (mutex_t *mp)`
  Initializes a mutex_t structure.
- `void chMtxLock (mutex_t *mp)`
  Locks the specified mutex.
- `void chMtxLockS (mutex_t *mp)`
  Locks the specified mutex.
- `bool chMtxTryLock (mutex_t *mp)`
  Tries to lock a mutex.
- `bool chMtxTryLockS (mutex_t *mp)`
  Tries to lock a mutex.
- `void chMtxUnlock (mutex_t *mp)`
  Unlocks the specified mutex.
- `void chMtxUnlockS (mutex_t *mp)`
  Unlocks the specified mutex.
- `void 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.

6.27.2 Macro Definition Documentation
6.27.1 __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.
## 6.27.2.2 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

## 6.27.3 Typedef Documentation

### 6.27.3.1 mutex_t

typedef struct ch_mutex mutex_t

Type of a mutex structure.

## 6.27.4 Function Documentation

### 6.27.4.1 chMtxObjectInit()

```c
void chMtxObjectInit ( mutex_t * mp )
```

Initializes s 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
```

### 6.27.4.2 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.
Here is the call graph for this function:

6.27.4.3 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

| 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 for chMtxLockS](image)

### 6.27.4.4 chMtxTryLock()

```c
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:

```
6.27.4.5 chMtxTryLockS()

bool chMtxTryLockS (  
    mutex_t * mp  )

Tries to lock a mutex.
This function attempts to lock a mutex, if the mutex is already taken by another thread then the function exits without waiting.

Postcondition
The mutex is locked and inserted in the per-thread stack of owned mutexes.

Note
This function does not have any overhead related to the priority inheritance mechanism because it does not try to enter a sleep state.
### 6.27 Mutexes

**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:

![Call graph](image)

#### 6.27.4.6 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.
Parameters

| in | mp | pointer to the `mutex_t` structure |

Function Class:

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

Here is the call graph for this function:

```
6.27.4.7 chMtxUnlockS()

void chMtxUnlockS ( 
                    mutex_t * mp )

Unlocks the specified mutex.
```
Note
Mutexes must be unlocked in reverse lock order. Violating this rule will result in a panic if assertions are enabled.

Precondition
The invoking thread **must** have at least one owned mutex.

Postcondition
The mutex is unlocked and removed from the per-thread stack of owned mutexes.
This function does not reschedule so a call to a rescheduling function must be performed before unlocking the kernel.

Parameters
\[ \text{in } \ mp \quad \text{pointer to the mutex_t structure} \]

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:

6.27.4.8 **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:

6.27.4.9 chMtxUnlockAll()

void chMtxUnlockAll ( )

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:

6.27.4.10 chMtxQueueNotEmptyS()

static bool chMtxQueueNotEmptyS ( mutex_t ∗ mp ) [inline], [static]

Returns true if the mutex queue contains at least a waiting thread.

Parameters

| out | mp | pointer to a mutex_t structure |

ChibiOS/RT
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
  ch_queue_notempty
chDbgCheckClassS
    chSysHalt
```

6.27.4.11  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
```

Returns

The owner thread.

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:

```
chMtxGetOwnerI → chDbgCheckClassI → chSysHalt
```

### 6.27.4.12 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:

```
chMtxGetNextMutexX → chThdGetSelfX
```
6.28 Condition Variables

6.28.1 Detailed Description

This module implements the Condition Variables mechanism. Condition variables are an extensions to the mutex subsystem and cannot work alone.

Operation mode

The condition variable is a synchronization object meant to be used inside a zone protected by a mutex. Mutexes and condition variables together can implement a Monitor construct.

Precondition

In order to use the condition variable APIs the CH_CFG_USE_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_DEC(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 s 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 chCondBroadcast1 (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.
6.28 Condition Variables

6.28.2 Macro Definition Documentation

6.28.2.1 __CONDVAR_DATA

#define __CONDVAR_DATA(name) {__CH_QUEUE_DATA(name.queue)}

Data part of a static condition variable initializer.

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

Parameters

- `in` `name` the name of the condition variable

6.28.2.2 CONDVAR_DECL

#define CONDVAR_DECL(name) condition_variable_t name = __CONDVAR_DATA(name)

Static condition variable initializer.

Statically initialized condition variables require no explicit initialization using chCondInit().

Parameters

- `in` `name` the name of the condition variable

6.28.3 Typedef Documentation

6.28.3.1 condition_variable_t

typedef struct condition_variable condition_variable_t

condition_variable_t structure.

6.28.4 Function Documentation

ChibiOS/RT
### 6.28.4.1 chCondObjectInit()

```c
void chCondObjectInit (
    condition_variable_t * cp )
```

Initializes a `condition_variable_t` structure.

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>out cp</code></td>
<td>pointer to a <code>condition_variable_t</code> structure</td>
</tr>
</tbody>
</table>

**Function Class:**

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

Here is the call graph for this function:

![Call Graph]

### 6.28.4.2 chCondSignal()

```c
void chCondSignal (
    condition_variable_t * cp )
```

Signals one thread that is waiting on the condition variable.

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>in cp</code></td>
<td>pointer to the <code>condition_variable_t</code> structure</td>
</tr>
</tbody>
</table>
6.28 Condition Variables

Function Class:

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

Here is the call graph for this function:

6.28.4.3 chCondSignalI()

```c
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 |

ChibiOS/RT
Function Class:

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

Here is the call graph for this function:

![Call Graph](image)

### 6.28.4.4 chCondBroadcast()

```c
void chCondBroadcast ( condition_variable_t * cp )
```

Signals all threads that are waiting on the condition variable.

**Parameters**

<table>
<thead>
<tr>
<th><strong>in cp</strong></th>
<th>pointer to the condition_variable_t structure</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:

```
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:

```
<table>
<thead>
<tr>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>chCondBroadcast</td>
</tr>
<tr>
<td>ch_queue_fifo_remove</td>
</tr>
<tr>
<td>ch_queue_notempty</td>
</tr>
<tr>
<td>chDbgCheckClassI</td>
</tr>
<tr>
<td>chSchReadyI</td>
</tr>
<tr>
<td>chSysHalt</td>
</tr>
<tr>
<td>__sch_ready_behind</td>
</tr>
<tr>
<td>chSysNotifyInstance</td>
</tr>
</tbody>
</table>
```

### 6.28.4.6 chCondWait()

```c
msg_t chCondWait (condition_variable_t * cp )
```

Waits on the condition variable releasing the mutex lock.

Releases the currently owned mutex, waits on the condition variable, and finally acquires the mutex again. All the sequence is performed atomically.

**Precondition**

The invoking thread **must** have at least one owned mutex.

**Parameters**

- **in 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()`.
Function Class:

Normal API, 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]

### 6.28.4.7 chCondWaitS()

```c
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>
<tbody>
<tr>
<td>MSG_RESET</td>
<td>if the condition variable has been signaled using chCondBroadcast().</td>
</tr>
</tbody>
</table>

Function Class:

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

Here is the call graph for this function:

![Call Graph](image)

### 6.28.4.8 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_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.
6.28 Condition Variables

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

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>MSG_OK</code></td>
<td>If the condition variable has been signaled using <code>chCondSignal()</code>.</td>
</tr>
<tr>
<td><code>MSG_RESET</code></td>
<td>If the condition variable has been signaled using <code>chCondBroadcast()</code>.</td>
</tr>
<tr>
<td><code>MSG_TIMEOUT</code></td>
<td>If the condition variable has not been signaled 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:
6.28.4.9 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**

- **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:

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

Here is the call graph for this function:
6.29 Event Flags

6.29.1 Detailed Description

Event Flags, Event Sources and Event Listeners.

Operation mode

Each thread has a mask of pending events inside its thread_t structure. Operations defined for events:

- **Wait**, the invoking thread goes to sleep until a certain AND/OR combination of events are signaled.
- **Clear**, a mask of events is cleared from the pending events, the cleared events mask is returned (only the events that were actually pending and then cleared).
- **Signal**, an events mask is directly ORed to the mask of the signaled thread.
- **Broadcast**, each thread registered on an Event Source is signaled with the events specified in its Event Listener.
- **Dispatch**, an events mask is scanned and for each bit set to one an associated handler function is invoked. Bit masks are scanned from bit zero upward.

An Event Source is a special object that can be "broadcasted" by a thread or an interrupt service routine. Broadcasting an Event Source has the effect that all the threads registered on the Event Source will be signaled with an events mask.
An unlimited number of Event Sources can exists in a system and each thread can be listening on an unlimited number of them.

Precondition

In order to use the Events APIs the CH_CFG_USE_EVENTS option must be enabled in chconf.h.

Postcondition

Enabling events requires 1-4 (depending on the architecture) extra bytes in the thread_t structure.

Macros

- `#define ALL_EVENTS ((eventmask_t)-1)`
  All events allowed mask.
- `#define EVENT_MASK(eid) ((eventmask_t)(1 << (eventmask_t)(eid)))`
  Returns an event mask from an event identifier.
- `#define __EVENTSOURCE_DATA(name) {(event_listener_t *)&name}`
  Data part of a static event source initializer.
- `#define EVENTSOURCEDECL(name) event_source_t name = __EVENTSOURCE_DATA(name)`
  Static event source initializer.
6.29 Event Flags

Typedefs

- typedef struct event_source event_source_t
  Event Source structure.
- typedef void(∗ evhandler_t)(eventid_t id)
  Event Handler callback function.

Data Structures

- struct event_listener
  Event Listener structure.
- struct event_source
  Event Source structure.

Functions

- void chEvtRegisterMaskWithFlagsI(event_source_t ∗esp, event_listener_t ∗elp, eventmask_t events, eventflags_t wflags)
  Registers an Event Listener on an Event Source.
- void chEvtRegisterMaskWithFlags(event_source_t ∗esp, event_listener_t ∗elp, eventmask_t events, eventflags_t wflags)
  Registers an Event Listener on an Event Source.
- void chEvtUnregister(event_source_t ∗esp, event_listener_t ∗elp)
  Unregisters an Event Listener from its Event Source.
- eventmask_t chEvtGetAndClearEventsI(eventmask_t events)
  Clears the pending events specified in the events mask.
- eventmask_t chEvtGetAndClearEvents(eventmask_t events)
  Clears the pending events specified in the events mask.
- eventmask_t chEvtAddEvents(eventmask_t events)
  Adds (OR) a set of events to the current thread, this is much faster than using chEvtBroadcast() or chEvtSignal().
- eventflags_t chEvtGetAndClearFlagsI(event_listener_t ∗elp)
  Returns the unmasked flags associated to an event_listener_t.
- eventflags_t chEvtGetAndClearFlags(event_listener_t ∗elp)
  Returns the flags associated to an event_listener_t.
- void chEvtSignalI(thread_t ∗tp, eventmask_t events)
  Adds a set of event flags directly to the specified thread_t.
- void chEvtSignal(thread_t ∗tp, eventmask_t events)
  Adds a set of event flags directly to the specified thread_t.
- void chEvtBroadcastFlagsI(event_source_t ∗esp, eventflags_t flags)
  Signals all the Event Listeners registered on the specified Event Source.
- void chEvtBroadcastFlags(event_source_t ∗esp, eventflags_t flags)
  Signals all the Event Listeners registered on the specified Event Source.
- void chEvt Dispatch(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.

### 6.29.2 Macro Definition Documentation

#### 6.29.2.1 ALL_EVENTS

```c
#define ALL_EVENTS ((eventmask_t)-1)
```

All events allowed mask.

#### 6.29.2.2 EVENT_MASK

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

Returns an event mask from an event identifier.

#### 6.29.2.3 __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 |

6.29.2.4 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 |

6.29.3 Typedef Documentation

6.29.3.1 event_source_t

typedef struct event_source event_source_t

Event Source structure.

6.29.3.2 evhandler_t

typedef void(* evhandler_t) (eventid_t id)

Event Handler callback function.

6.29.4 Function Documentation
6.29.4.1 chEvtRegisterMaskWithFlagsI()

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

Registers an Event Listener on an Event Source.

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

**Note**

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

**Parameters**

<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:**

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

Here is the call graph for this function:

![Call Graph](image)

6.29.4.2 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>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>esp</td>
<td>pointer to the event_source_t structure</td>
</tr>
<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:

```
6.29.4.3 chEvtUnregister()

void chEvtUnregister (  
        event_source_t * esp,  
        event_listener_t * elp )

Unregisters an Event Listener from its Event Source.

Note

If the event listener is not registered on the specified event source then the function does nothing.  
For optimal performance it is better to perform the unregister operations in inverse order of the register operations (elements are found on top of the list). 

ChibiOS/RT
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>eIp</td>
<td>pointer to the event_listener_t structure</td>
</tr>
</tbody>
</table>

Function Class:

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

Here is the call graph for this function:

6.29.4.4 chEvtGetAndClearEventsI()

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

Clears the pending events specified in the events mask.

Parameters

| in | events | the events to be cleared |

Returns

The mask of pending events that were cleared.

Function Class:

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

![Call Graph Diagram]

### 6.29.4.5 chEvtGetAndClearEvents()

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

Clears the pending events specified in the events mask.

**Parameters**

- **events**
  - the events to be cleared

**Returns**

- The mask of pending events that were cleared.
### Function Class:

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

Here is the call graph for this function:

![Call Graph]

### 6.29.4.6 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:

6.29.4.7 chEvtGetAndClearFlagsI()

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

Returns the unmasked flags associated to an event_listener_t.

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

Parameters

| in | 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.

Here is the call graph for this function:

```
chEvtGetAndClearFlagsI —> chDbgCheckClassI —> chSysHalt
```

6.29.4.8  **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:

6.29.4.9 chEvtSignalI()

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

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

Postcondition

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

Parameters

<table>
<thead>
<tr>
<th></th>
<th>tp</th>
<th>the thread to be signaled</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>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:

```
chEvtSignalI
chDbgCheckClassI
chSchReadyI
chSysHalt
__sch_ready_behind
chSysNotifyInstance
```

### 6.29.4.10 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>in</th>
<th>tp</th>
<th>the thread to be signaled</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>events</td>
<td>the events set to be ORed</td>
</tr>
</tbody>
</table>

Function Class:

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

void chEvtBroadcastFlagsI (  
    event_source_t * esp,  
    eventflags_t flags )

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

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

Postcondition

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

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in esp</td>
<td>pointer to the event_source_t structure</td>
</tr>
<tr>
<td>in 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:

### 6.29.4.12 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>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>esp</code></td>
<td>pointer to the <code>event_source_t</code> structure</td>
</tr>
<tr>
<td><code>flags</code></td>
<td>the flags set to be added to the listener flags mask</td>
</tr>
</tbody>
</table>

**Function Class:**

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

Here is the call graph for this function:
6.29.4.13  chEvtDispatch()

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

Invokes the event handlers associated to an event flags mask.

**Parameters**

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

6.29.4.14  chEvtWaitOne()

```c
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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>events</code></td>
<td>events that the function should wait for, <code>ALL_EVENTS</code> enables all the events</td>
</tr>
</tbody>
</table>

**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:

6.29.4.15 chEvtWaitAny()

```
eventmask_t chEvtWaitAny (  
    eventmask_t events )
```

Waits for any of the specified events.

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

Parameters

| in  | 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:

![Call Graph Diagram]

6.29.4.16 chEvtWaitAll()

```c
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:

![Call Graph](image)

6.29.4.17 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 an higher priority.

Parameters

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

ChibiOS/RT
6.29 Event Flags

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:

```
6.29.4.18 chEvtWaitAnyTimeout()

eventmask_t chEvtWaitAnyTimeout (  
  eventmask_t events,  
  sysinterval_t timeout )

Waits for any of the specified events.

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

Parameters

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

ChibiOS-RT
Returns

The mask of the served and cleared events.

Return values

0 if the operation has timed out.

Function Class:

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

Here is the call graph for this function:

```
6.29.4.19 chEvtWaitAllTimeout()

eventmask_t chEvtWaitAllTimeout (
    eventmask_t events,
    sysinterval_t timeout )

Waits for all the specified events.

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

Parameters

<table>
<thead>
<tr>
<th></th>
<th>events</th>
<th>events that the function should wait for, ALL_EVENTS requires 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_INFINITY no timeout.</td>
</tr>
</tbody>
</table>
```
6.29 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:

6.29.4.20 chEvtObjectInit()

static void chEvtObjectInit ( event_source_t * esp ) [inline], [static]

Initializes an Event Source.

Note

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

Parameters

in esp pointer to the event_source_t structure
Function Class:

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

### 6.29.4.21 chEvtRegisterMask()

```c
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>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td><code>esp</code> pointer to the <code>event_source_t</code> structure</td>
</tr>
<tr>
<td>out</td>
<td><code>elp</code> pointer to the <code>event_listener_t</code> structure</td>
</tr>
<tr>
<td>in</td>
<td><code>events</code> 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:

```plaintext
chEvtRegisterMask chEvtRegisterMaskWithFlags chEvtRegisterMaskWithFlagsI chSysLock chSysUnlock
```

### 6.29.4.22 chEvtRegister()

```c
static void chEvtRegister (  
    event_source_t * esp,
```
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:

6.29.4.23 chEvtIsListeningI()

Verifies if there is at least one `event_listener_t` registered.

Parameters

| in  | esp | pointer to the `event_source_t` structure |

Returns
The event source status.
Function Class:

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

### 6.29.4.24 chEvtBroadcast()

```c
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:

![Call Graph](image)

### 6.29.4.25 chEvtBroadcastI()

```c
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
chEvtSignal

6.29.4.26 chEvtAddEventsI()

static eventmask_t chEvtAddEventsI (eventmask_t events) [inline], [static]

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

Parameters

in events the events to be added

Returns

The mask of currently pending events.

Function Class:

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

6.29.4.27 chEvtGetEventsX()

static eventmask_t chEvtGetEventsX (void ) [inline], [static]

Returns the events mask.

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

The pending events mask.

Function Class:

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

6.30.1 Detailed Description

Synchronous inter-thread messages APIs and services.

Operation Mode

Synchronous messages are an easy to use and fast IPC mechanism, threads can both act as message servers and/or message clients, the mechanism allows data to be carried in both directions. Note that messages are not copied between the client and server threads but just a pointer passed so the exchange is very time efficient. Messages are scalar data types of type `msg_t` that are guaranteed to be size compatible with data pointers. Note that on some architectures function pointers can be larger 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 * 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.
6.30.2 Function Documentation

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

**Returns**

The answer message from `chMsgRelease()`.

**Function Class:**

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

Here is the call graph for this function:

6.30.2.2 chMsgWaitS()

```c
thread_t * chMsgWaitS ( 
    void )
```

Suspends the thread and waits for an incoming message.
6.30 Synchronous Messages

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:

6.30.2.3 `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>• 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:

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

Here is the call graph for this function:

```
6.30.2.4 chMsgPollS()

thread_t * chMsgPollS (  
    void  )  
```

Poll to check for an incoming message.

Postcondition

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

Note

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

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

Returns

Result of the poll.
6.30 Synchronous Messages

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:

![Call Graph](call_graph.png)

6.30.2.5 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]

6.30.2.6 chMsgWait()

```c
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.
6.30 Synchronous Messages

Function Class:

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

Here is the call graph for this function:

![Call Graph](image)

6.30.2.7 chMsgWaitTimeout()

```c
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>• <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

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:

6.30.2.8 chMsgPoll()

```c
static thread_t* chMsgPoll ( void ) [inline], [static]
```

Poll to check for an incoming message.
Postcondition

If a message is available the function \texttt{chMsgGet()} must be called in order to retrieve the message and then \texttt{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 \texttt{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

\texttt{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:
6.30.2.9 chMsgIsPendingI()

static bool chMsgIsPendingI (  
    thread_t * tp ) [inline], [static]

Evaluates to true if the thread has pending messages.

Parameters

| in  | 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:

6.30.2.10 chMsgGet()

static msg_t chMsgGet (  
    thread_t * tp ) [inline], [static]

Returns the message carried by the specified thread.

Precondition

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

Parameters

| in  | tp  | pointer to the thread |
Returns

The message carried by the sender.

Function Class:

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

6.30.2.11  chMsgReleaseS()

static void chMsgReleaseS (  
   thread_t  * tp,  
   msg_t  msg )  [inline], [static]

Releases the thread waiting on top of the messages queue.

Precondition

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

Parameters

| in  | tp | pointer to the thread |
| in  | 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:
6.31 Dynamic Threads

6.31.1 Detailed Description

Dynamic threads related APIs and services.

Functions

- `thread_t * chThdCreateFromHeap (memory_heap_t *heapp, size_t size, const char *name, tprio_t prio, tfunc_t pf, void *arg)`
  
  Creates a new thread allocating the memory from the heap.

- `thread_t * chThdCreateFromMemoryPool (memory_pool_t *mp, const char *name, tprio_t prio, tfunc_t pf, void *arg)`
  
  Creates a new thread allocating the memory from the specified memory pool.

6.31.2 Function Documentation

6.31.2.1 chThdCreateFromHeap()

```c
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><code>heapp</code></td>
<td>heap from which allocate the memory or NULL for the default heap</td>
</tr>
<tr>
<td><code>size</code></td>
<td>size of the working area to be allocated</td>
</tr>
<tr>
<td><code>name</code></td>
<td>thread name</td>
</tr>
<tr>
<td><code>prio</code></td>
<td>the priority level for the new thread</td>
</tr>
<tr>
<td><code>pf</code></td>
<td>the thread function</td>
</tr>
<tr>
<td><code>arg</code></td>
<td>an argument passed to the thread function. It can be NULL.</td>
</tr>
</tbody>
</table>
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:

6.31.2.2 chThdCreateFromMemoryPool()

thread_t * chThdCreateFromMemoryPool (  
  memory_pool_t * mp,  
  const char * name,  
  tprio_t prio,  
  tfunc_t pf,  
  void * arg )

Creates a new thread allocating the memory from the specified memory pool.
Precondition

The configuration options `CH_CFG_USE_DYNAMIC` and `CH_CFG_USE_MEMPOOLS` must be enabled in order to use this function.

The pool must be initialized to contain only objects with alignment `PORT_WORKING_AREA_ALIGN`.

Note

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

The memory allocated for the thread is not released automatically, it is responsibility of the creator thread to call `chThdWait()` and then release the allocated memory.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>mp</th>
<th>pointer to the memory pool object</th>
</tr>
</thead>
<tbody>
<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>
<tr>
<td>in</td>
<td>pf</td>
<td>the thread function</td>
</tr>
<tr>
<td>in</td>
<td>arg</td>
<td>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.

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:
6.32 Registry

6.32.1 Detailed Description

Threads Registry related APIs and services.

Operation mode

The Threads Registry is a double linked list that holds all the active threads in the system. Operations defined for the registry:

- **First**, returns the first, in creation order, active thread in the system.
- **Next**, returns the next, in creation order, active thread in the system.

The registry is meant to be mainly a debug feature, for example, using the registry a debugger can enumerate the active threads in any given moment or the shell can print the active threads and their state. Another possible use is for centralized threads memory management, terminating threads can pulse an event source and an event handler can perform a scansion of the registry in order to recover the memory.

Precondition

In order to use the threads registry the `CH_CFG_USE_REGISTRY` option must be enabled in `chconf.h`.

Macros

- `#define REG_HEADER(oip) (&ch_system.reglist.queue)`
  Access to the registry list header.
- `#define REG_REMOVE(tp) (void) ch_queue_dequeue(&(tp)->rqueue)`
  Removes a thread from the registry list.
- `#define REG_INSERT(oip, tp) ch_queue_insert(REG_HEADER(oip), &(tp)->rqueue)`
  Adds a thread to the registry list.

Data Structures

- `struct chdebug_t`
  ChibiOS/RT memory signature record.
Functions

- `thread_t * chRegFirstThread (void)`
  Returns the first thread in the system.

- `thread_t * chRegNextThread (thread_t *tp)`
  Returns the thread next to the specified one.

- `thread_t * chRegFindThreadByName (const char *name)`
  Retrieves a thread pointer by name.

- `thread_t * chRegFindThreadByPointer (thread_t *tp)`
  Confirms that a pointer is a valid thread pointer.

- `thread_t * chRegFindThreadByWorkingArea (stkalign_t *wa)`
  Confirms that a working area is being used by some active thread.

- `static void __reg_object_init (registry_t *rp)`
  Initializes a registry.

- `static void chRegSetThreadName (const char *name)`
  Sets the current thread name.

- `static const char * chRegGetThreadNameX (thread_t *tp)`
  Returns the name of the specified thread.

- `static void chRegSetThreadNameX (thread_t *tp, const char *name)`
  Changes the name of the specified thread.

6.32.2 Macro Definition Documentation

6.32.2.1 REG_HEADER

```c
#define REG_HEADER(oip) (&ch_system.reglist.queue)
```
Access to the registry list header.

6.32.2.2 REG_REMOVE

```c
#define REG_REMOVE(tp) (void) ch_queue_dequeue(&(tp)->rqueue)
```
Removes a thread from the registry list.

Note
This macro is not meant for use in application code.

Parameters

| in  | tp | thread to remove from the registry |
6.32.2.3 REG_INSERT

```c
#define REG_INSERT(
    oip,
    tp ) ch_queue_insert(REG_HEADER(oip), &(tp)->rqueue)
```

Adds a thread to the registry list.

Note

This macro is not meant for use in application code.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>oip</th>
<th>pointer to the OS instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>tp</td>
<td>thread to add to the registry</td>
</tr>
</tbody>
</table>

6.32.3 Function Documentation

6.32.3.1 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:

6.32.3.2 chRegNextThread()

thread_t * chRegNextThread ( thread_t * tp )

Returns the thread next to the specified one.

The reference counter of the specified thread is decremented and the reference counter of the returned thread is incremented.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>tp</th>
<th>pointer to the thread</th>
</tr>
</thead>
</table>

Returns

A reference to the next thread.

Return values

NULL if there is no next thread.

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

6.32.3.3 chRegFindThreadByName()

```
thread_t * chRegFindThreadByName (  
    const char * name )
```

Retrieves a thread pointer by name.

Note

The reference counter of the found thread is increased by one so it cannot be disposed incidentally after the pointer has been returned.

Parameters

| in   | name | the thread name |

Returns

A pointer to the found thread.

Return values

| NULL | if a matching thread has not been found. |
Function Class:
Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

```
6.32.3.4 chRegFindThreadByPointer()

thread_t * chRegFindThreadByPointer ( thread_t * tp )
```

Confirms that a pointer is a valid thread pointer.

Note
The reference counter of the found thread is increased by one so it cannot be disposed incidentally after the pointer has been returned.

Parameters

| in  | 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]

6.32.3.5 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:

---

6.32.3.6 __reg_object_init()

static void __reg_object_init (  
    registry_t * rp ) [inline], [static]

Initializes a registry.

Note

Internal use only.

Parameters

* out rp pointer to a registry_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:
6.32.3.7 chRegSetThreadName()

static void chRegSetThreadName (  
  const char * name ) [inline], [static]

Sets the current thread name.

Precondition

This function only stores the pointer to the name if the option CH_CFG_USE_REGISTRY is enabled else no action is performed.

Parameters

  in  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.

6.32.3.8 chRegGetThreadNameX()

static const char* chRegGetThreadNameX (  
  thread_t * tp ) [inline], [static]

Returns the name of the specified thread.

Precondition

This function only returns the pointer to the name if the option CH_CFG_USE_REGISTRY is enabled else NULL is returned.

Parameters

  in  tp  pointer to the thread

Returns

Thread name as a zero terminated string.

Return values

  NULL  if the thread name has not been set.
6.32.3.9  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.
6.33 Debug

6.33.1 Detailed Description

Modules

- Checks and Assertions
- Tracing
- Statistics
6.34 Checks and Assertions

6.34.1 Detailed Description

Debug APIs and services:

- Runtime system state and call protocol check. The following panic messages can be generated:
  - SV#1, misplaced `chSysDisable()`.  
    - Called from an ISR.  
    - Called from a critical zone.
  - SV#2, misplaced `chSysSuspend()`.  
    - Called from an ISR.  
    - Called from a critical zone.
  - SV#3, misplaced `chSysEnable()`.  
    - Called from an ISR.  
    - Called from a critical zone.
  - SV#4, misplaced `chSysLock()`.  
    - Called from an ISR.  
    - Called from a critical zone.
  - SV#5, misplaced `chSysUnlock()`.  
    - Called from an ISR.  
    - Not called from a critical zone.
  - SV#6, misplaced `chSysLockFromISR()`.  
    - Not called from an ISR.  
    - Called from a critical zone.
  - SV#7, misplaced `chSysUnlockFromISR()`.  
    - Not called from an ISR.  
    - Not called from a critical zone.
  - SV#8, misplaced `CH_IRQ_PROLOGUE()`.  
    - Not called at ISR begin.  
    - Called from a critical zone.
  - SV#9, misplaced `CH_IRQ_EPILOGUE()`.  
    - `CH_IRQ_PROLOGUE()` missing.  
    - Not called at ISR end.  
    - Called from a critical zone.
  - SV#10, misplaced I-class function.  
    - I-class function not called from within a critical zone.
  - SV#11, misplaced S-class function.  
    - S-class function not called from within a critical zone.  
    - Called from an ISR.

- Parameters check.

- Kernel assertions.

Note

Stack checks are not implemented in this module but in the port layer in an architecture-dependent way.
Debug related settings

- \#define CH_DBG_STACK_FILL_VALUE 0x55
  Fill value for thread stack area in debug mode.

Macro Functions

- \#define chDbgCheck(c)
  Function parameters check.
- \#define chDbgAssert(c, r)
  Condition assertion.

Typedefs

- typedef struct ch_system_debug system_debug_t
  System debug data structure.

Data Structures

- struct ch_system_debug
  System debug data structure.

Functions

- void __dbg_check_disable (void)
  Guard code for chSysDisable().
- void __dbg_check_suspend (void)
  Guard code for chSysSuspend().
- void __dbg_check_enable (void)
  Guard code for chSysEnable().
- void __dbg_check_lock (void)
  Guard code for chSysLock().
- void __dbg_check_unlock (void)
  Guard code for chSysUnlock().
- void __dbg_check_lock_from_isr (void)
  Guard code for chSysLockFromIsr().
- void __dbg_check_unlock_from_isr (void)
  Guard code for chSysUnlockFromIsr().
- void __dbg_check_enter_isr (void)
  Guard code for CH_IRQ_PROLOGUE().
- void __dbg_check_leave_isr (void)
  Guard code for CH_IRQ_EPILOGUE().
- void chDbgCheckClassI (void)
  I-class functions context check.
- void chDbgCheckClassS (void)
  S-class functions context check.
- static void __dbg_object_init (system_debug_t *sdp)
  Debug support initialization.
6.34.2 Macro Definition Documentation

6.34.2.1 CH_DBG_STACK_FILL_VALUE

#define CH_DBG_STACK_FILL_VALUE 0x55

Fill value for thread stack area in debug mode.

6.34.2.2 chDbgCheck

#define chDbgCheck(
	c
)

Value:

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

Function parameters check.

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

Note

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

Parameters

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

Function Class:

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

6.34.2.3 chDbgAssert

#define chDbgAssert(
	c,
	r
)
Value:

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

Condition assertion.

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

Note

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

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

Parameters

| in | c     | the condition to be verified to be true |
|    | r     | a remark string                         |

Function Class:

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

6.34.3 Typedef Documentation

6.34.3.1 system_debug_t

typedef struct ch_system_debug system_debug_t

System debug data structure.

6.34.4 Function Documentation
6.34.4.1 __dbg_check_disable()

void __dbg_check_disable (  
                              void )

Guard code for chSysDisable().

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

Here is the call graph for this function:

```
__dbg_check_disable -> chSysHalt -> __trace_halt
```

6.34.4.2 __dbg_check_suspend()

void __dbg_check_suspend (  
                              void )

Guard code for chSysSuspend().

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

Here is the call graph for this function:

```
__dbg_check_suspend -> chSysHalt -> __trace_halt
```
6.34.4.3  __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:

```
__dbg_check_enable   chSysHalt   __trace_halt
```

6.34.4.4  __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
```
6.34.4.5  __dbg_check_unlock()

void __dbg_check_unlock 
  
Guard code for chSysUnlock().

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

Here is the call graph for this function:

6.34.4.6  __dbg_check_lock_from_isr()

void __dbg_check_lock_from_isr 

Guard code for chSysLockFromIsr().

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

Here is the call graph for this function:
6.34.4.7 __dbg_check_unlock_from_isr()

```c
void __dbg_check_unlock_from_isr ( void )
```

Guard code for chSysUnlockFromISR().

Function Class:

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

Here is the call graph for this function:

![Call Graph](chart.png)

6.34.4.8 __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:

![Call Graph](chart.png)
6.34.4.9 __dbg_check_leave_isr()

```c
void __dbg_check_leave_isr ( void )
```

Guard code for CH_IRQ_EPILOGUE().

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

Here is the call graph for this function:

```
| __dbg_check_leave_isr | chSysHalt | __trace_halt |
```

6.34.4.10 chDbgCheckClassI()

```c
void chDbgCheckClassI ( void )
```

I-class functions context check.

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

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

Here is the call graph for this function:

```
| chDbgCheckClassI | chSysHalt | __trace_halt |
```
6.34.4.11  chDbgCheckClassS()

void chDbgCheckClassS ( void )

S-class functions context check.

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

Function Class:

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

Here is the call graph for this function:

```
  chDbgCheckClassS → chSysHalt → __trace_halt
```

6.34.4.12  __dbg_object_init()

static void __dbg_object_init ( system_debug_t * sdp ) [inline], [static]

Debug support initialization.

Note

Internal use only.

Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>sdp pointer to the system_debug_t structure</td>
</tr>
</tbody>
</table>

Function Class:

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

6.35.1 Detailed Description

System events tracing service.

Trace record types

- \#define CH_TRACE_TYPE_UNUSED 0U
- \#define CH_TRACE_TYPE_READY 1U
- \#define CH_TRACE_TYPE_SWITCH 2U
- \#define CH_TRACE_TYPE_ISR_ENTER 3U
- \#define CH_TRACE_TYPE_ISR_LEAVE 4U
- \#define CH_TRACE_TYPE_HALT 5U
- \#define CH_TRACE_TYPE_USER 6U

Events to trace

- \#define CH_DBG_TRACE_MASK_DISABLED 255U
- \#define CH_DBG_TRACE_MASK_NONE 0U
- \#define CH_DBG_TRACE_MASK_READY 1U
- \#define CH_DBG_TRACE_MASK_SWITCH 2U
- \#define CH_DBG_TRACE_MASK_ISR 4U
- \#define CH_DBG_TRACE_MASK_HALT 8U
- \#define CH_DBG_TRACE_MASK_USER 16U
- \#define CH_DBG_TRACE_MASK_SLOW
- \#define CH_DBG_TRACE_MASK_ALL

Debug related settings

- \#define CH_DBG_TRACE_MASK \#define CH_DBG_TRACE_MASK_DISABLED
  Trace buffer entries.
- \#define CH_DBG_TRACE_BUFFER_SIZE 128
  Trace buffer entries.

Data Structures

- struct trace_event_t
  Trace buffer record.
- struct trace_buffer_t
  Trace buffer header.
6.35 Tracing

Functions

- static NOINLINE void trace_next (os_instance_t *oip)
  Writes a time stamp and increases the trace buffer pointer.
- void __trace_object_init (trace_buffer_t *tbp)
  Circular trace buffer initialization.
- void __trace_ready (thread_t *tp, msg_t msg)
  Inserts in the circular debug trace buffer a ready record.
- void __trace_switch (thread_t *ntp, thread_t *otp)
  Inserts in the circular debug trace buffer a context switch record.
- void __trace_isr_enter (const char *isr)
  Inserts in the circular debug trace buffer an ISR-enter record.
- void __trace_isr_leave (const char *isr)
  Inserts in the circular debug trace buffer an ISR-leave record.
- void __trace_halt (const char *reason)
  Inserts in the circular debug trace buffer an halt record.
- void chTraceWriteI (void *up1, void *up2)
  Adds an user trace record to the trace buffer.
- void chTraceWrite (void *up1, void *up2)
  Adds an user trace record to the trace buffer.
- void chTraceSuspendI (uint16_t mask)
  Suspends one or more trace events.
- void chTraceSuspend (uint16_t mask)
  Suspends one or more trace events.
- void chTraceResumeI (uint16_t mask)
  Resumes one or more trace events.
- void chTraceResume (uint16_t mask)
  Resumes one or more trace events.

6.35.2 Macro Definition Documentation

6.35.2.1 CH_DBG_TRACE_MASK

#define CH_DBG_TRACE_MASK CH_DBG_TRACE_MASK_DISABLED

Trace buffer entries.

6.35.2.2 CH_DBG_TRACE_BUFFER_SIZE

#define CH_DBG_TRACE_BUFFER_SIZE 128

Trace buffer entries.

Note

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

6.35.3.1 trace_next()

```c
static NONINLINE void trace_next (  
   os_instance_t * oip ) [static]
```

Writes a time stamp and increases the trace buffer pointer.

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

Here is the call graph for this function:

![Call Graph](image)

6.35.3.2 __trace_object_init()

```c
void __trace_object_init (  
   trace_buffer_t * tbp )
```

Circular trace buffer initialization.

Note
Internal use only.

Parameters

- `out tbp` pointer to the `trace_buffer_t` structure

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

void __trace_ready (  
    thread_t * tp,  
    msg_t msg )

Inserts in the circular debug trace buffer a ready record.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>tp</th>
<th>the thread that just become ready</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>msg</td>
<td>the thread ready message</td>
</tr>
</tbody>
</table>

Function Class:

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

6.35.3.4  __trace_switch()

void __trace_switch (  
    thread_t * ntp,  
    thread_t * otp )

Inserts in the circular debug trace buffer a context switch record.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>ntp</th>
<th>the thread being switched in</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</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.

6.35.3.5  __trace_isr_enter()

void __trace_isr_enter (  
    const char * isr )

Inserts in the circular debug trace buffer an ISR-enter record.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>isr</th>
<th>name of the isr</th>
</tr>
</thead>
</table>
Function Class:

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

6.35.3.6 __trace_isr_leave()

```c
void __trace_isr_leave ( const char * isr )
```

Inserts in the circular debug trace buffer an ISR-leave record.

Parameters

| in | isr | name of the isr |

Function Class:

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

6.35.3.7 __trace_halt()

```c
void __trace_halt ( const char * reason )
```

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.

6.35.3.8 chTraceWriteI()

```c
void chTraceWriteI ( 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]

6.35.3.9 **chTraceWrite()**

```c
void chTraceWrite ( 
    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:

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

Here is the call graph for this function:

```
6.35.3.10  chTraceSuspendI()

void chTraceSuspendI (  
   uint16_t mask 
)

Suspends one or more trace events.

Parameters

| in  | mask | mask of the trace events to be suspended |
```
Function Class:

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

Here is the call graph for this function:

6.35.3.11 chTraceSuspend()

```c
void chTraceSuspend (  
    uint16_t mask )
```

Suspends one or more trace events.

Parameters

| in   | mask | mask of the trace events to be suspended |

Function Class:

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

void chTraceResumeI (  
  uint16_t mask  )

Resumes one or more trace events.

Parameters

  \textbf{in} \quad \textit{mask} \quad \text{mask of the trace events to be resumed}

Function Class:

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

Here is the call graph for this function:
6.35.3.13  chTraceResume()

void chTraceResume ( 
    uint16_t mask )

Resumes one or more trace events.

Parameters

|   | 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:
6.36  Statistics

6.36.1  Detailed Description

Statistics services.

Data Structures

- struct kernel_stats_t
  
  Type of a kernel statistics structure.

Functions

- void __stats_increase_irq (void)
  
  Increases the IRQ counter.

- void __stats_ctxswc (thread_t *ntp, thread_t *otp)
  
  Updates context switch related statistics.

- void __stats_start_measure_crit_thd (void)
  
  Starts the measurement of a thread critical zone.

- void __stats_stop_measure_crit_thd (void)
  
  Stops the measurement of a thread critical zone.

- void __stats_start_measure_crit_isr (void)
  
  Starts the measurement of an ISR critical zone.

- void __stats_stop_measure_crit_isr (void)
  
  Stops the measurement of an ISR critical zone.

- static void __stats_object_init (kernel_stats_t *ksp)
  
  Statistics initialization.

6.36.2  Function Documentation

6.36.2.1  __stats_increase_irq()

void __stats_increase_irq ( void )

Increases the IRQ counter.

6.36.2.2  __stats_ctxswc()

void __stats_ctxswc ( thread_t * ntp,
    thread_t * otp )

Updates context switch related statistics.
Parameters

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

Here is the call graph for this function:

```
__stats_ctxswc chTMChainMeasurementToX
```

### 6.36.2.3 __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 chTMStartMeasurementX
```

### 6.36.2.4 __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:

```
void __stats_start_measure_crit_isr ()
{
    void
}

Starts the measurement of an ISR critical zone.

Here is the call graph for this function:

```
void __stats_stop_measure_crit_isr ()
{
    void
}

Stops the measurement of an ISR critical zone.

Here is the call graph for this function:
6.36 Statistics

6.36.2.7 __stats_object_init()

static void __stats_object_init (kernel_stats_t * ksp) [inline], [static]

Statistics initialization.

Note

Internal use only.

Parameters

| out ksp | pointer to the kernel__stats_t structure |

Function Class:

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

Here is the call graph for this function:

__stats_object_init → chTMObjectInit
6.37 OS Library

6.37.1 Detailed Description

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

Modules

- Version Numbers and Identification
- Synchronization
- Memory Management
- Complex Services
6.38 Version Numbers and Identification

6.38.1 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 "1.3.0"
  
  OS Library version string.

- \#define CH\_OSLIB\_MAJOR 1
  
  OS Library version major number.

- \#define CH\_OSLIB\_MINOR 3
  
  OS Library version minor number.

- \#define CH\_OSLIB\_PATCH 0
  
  OS Library version patch number.

Functions

- static void \_oslib\_init (void)
  
  Initialization of all library modules.

6.38.2 Macro Definition Documentation

6.38.2.1 \_CHIBIOS\_OSLIB\_

\#define \_CHIBIOS\_OSLIB\_

ChibiOS/LIB identification macro.

6.38.2.2 CH\_OSLIB\_STABLE

\#define CH\_OSLIB\_STABLE 1

Stable release flag.
6.38.2.3 CH_OSLIB_VERSION

#define CH_OSLIB_VERSION "1.3.0"

OS Library version string.

6.38.2.4 CH_OSLIB_MAJOR

#define CH_OSLIB_MAJOR 1

OS Library version major number.

6.38.2.5 CH_OSLIB_MINOR

#define CH_OSLIB_MINOR 3

OS Library version minor number.

6.38.2.6 CH_OSLIB_PATCH

#define CH_OSLIB_PATCH 0

OS Library version patch number.

6.38.3 Function Documentation
### 6.38.3.1 __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:
6.39  Synchronization

6.39.1  Detailed Description

Synchronization services.

Modules

• Binary Semaphores
• Mailboxes
• Pipes
• Delegate Threads
• Jobs Queues
6.40 Binary Semaphores

6.40.1 Detailed Description

Macros

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

Typedefs

- `typedef struct ch_binary_semaphore binary_semaphore_t
  Binary semaphore type.

Data Structures

- `struct ch_binary_semaphore
  Binary semaphore type.

Functions

- `static void chBSemObjectInit (binary_semaphore_t *bsp, bool taken)
  Initializes a binary semaphore.
- `static msg_t chBSemWait (binary_semaphore_t *bsp)
  Wait operation on the binary semaphore.
- `static msg_t chBSemWaitS (binary_semaphore_t *bsp)
  Wait operation on the binary semaphore.
- `static msg_t chBSemWaitTimeoutS (binary_semaphore_t *bsp, sysinterval_t timeout)
  Wait operation on the binary semaphore.
- `static msg_t chBSemWaitTimeout (binary_semaphore_t *bsp, sysinterval_t timeout)
  Wait operation on the binary semaphore.
- `static void chBSemResetI (binary_semaphore_t *bsp, bool taken)
  Reset operation on the binary semaphore.
- `static void chBSemReset (binary_semaphore_t *bsp, bool taken)
  Reset operation on the binary semaphore.
- `static void chBSemSignalI (binary_semaphore_t *bsp)
  Performs a signal operation on a binary semaphore.
- `static void chBSemSignal (binary_semaphore_t *bsp)
  Performs a signal operation on a binary semaphore.
- `static bool chBSemGetStateI (const binary_semaphore_t *bsp)
  Returns the binary semaphore current state.

6.40.2 Macro Definition Documentation
6.40.2.1 __BSEMAPHORE_DATA

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

Data part of a static semaphore initializer.

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

Parameters

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

6.40.2.2 BSEMAPHORE_DECL

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

Static semaphore initializer.

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

Parameters

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

6.40.3 Typedef Documentation

6.40.3.1 binary_semaphore_t

typedef struct ch_binary_semaphore binary_semaphore_t

Binary semaphore type.

6.40.4 Function Documentation
6.40 Binary Semaphores

6.40.4.1 chBSemObjectInit()

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

Initializes a binary semaphore.

Parameters

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

- false, the initial state is not taken.
- true, the initial state is taken.

Function Class:

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

Here is the call graph for this function:

6.40.4.2 chBSemWait()

static msg_t chBSemWait (
        binary_semaphore_t ∗ bsp ) [inline], [static]

Wait operation on the binary semaphore.

Parameters

| in | bsp | pointer to a binary_semaphore_t structure |

Returns

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

<table>
<thead>
<tr>
<th>Return Value</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:

6.40.4.3 chBSemWaitS()

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

Wait operation on the binary semaphore.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in bsp</td>
<td>pointer to a binary_semaphore_t 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>Return Value</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>
6.40 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
<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

### 6.40.4.4 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>bsp</th>
<th>pointer to a binary_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 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]

### 6.40.4.5 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 binary_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 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:

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

Here is the call graph for this function:

```
chBSemWaitTimeoutS

chBSemWaitTimeout

chSemWaitTimeout

chSysLock

chSysUnlock
```

### 6.40.4.6 chBSemResetI()

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

Reset operation on the binary semaphore.

**Note**

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

This function does not reschedule.

**Parameters**

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

- `false`, the new state is not taken.
- `true`, the new state is taken.
Function Class:

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

Here is the call graph for this function:

```
chBSemResetI chDbgCheckClassI
chSemResetI
chSysHalt
chSemResetWithMessageI
```

### 6.40.4.7 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>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:

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

Here is the call graph for this function:

```
chBSemReset chSemReset chSemResetWithMessage
```
6.40.4.8  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

\textbf{in} \hspace{1em} \textbf{bsp} \hspace{1em} pointer to a binary_semaphore_t structure

Function Class:

This is an \textbf{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.png)

6.40.4.9  chBSemSignal()

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

Performs a signal operation on a binary semaphore.

Parameters

\textbf{in} \hspace{1em} \textbf{bsp} \hspace{1em} pointer to a binary_semaphore_t structure
Function Class:
Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

6.40.4.10 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:
6.41 Mailboxes

6.41.1 Detailed Description

Asynchronous messages.

Operation mode

A mailbox is an asynchronous communication mechanism. Operations defined for mailboxes:

- **Post**: Posts a message on the mailbox in FIFO order.
- **Post Ahead**: Posts a message on the mailbox with urgent priority.
- **Fetch**: A message is fetched from the mailbox and removed from the queue.
- **Reset**: The mailbox is emptied and all the stored messages are lost.

A message is a variable of type msg_t that is guaranteed to have the same size of and be compatible with (data) pointers (anyway an explicit cast is needed). If larger messages need to be exchanged then a pointer to a structure can be posted in the mailbox but the posting side has no predefined way to know when the message has been processed. A possible approach is to allocate memory (from a memory pool for example) from the posting side and free it on the fetching side. Another approach is to set a "done" flag into the structure pointed by the message.

Precondition

In order to use the mailboxes APIs the CH_CFG_USE_MAILBOXES option must be enabled in chconf.h.

Note

Compatible with RT and NIL.

Macros

- `#define __MAILBOX_DATA(name, buffer, size)`
  
  Data part of a static mailbox initializer.

- `#define MAILBOX_DECL(name, buffer, size) mailbox_t name = __MAILBOX_DATA(name, buffer, size)`
  
  Static mailbox initializer.

Data Structures

- `struct mailbox_t`
  
  Structure representing a mailbox object.
Functions

- **void chMBObjectInit (mailbox_t *mbp, msg_t *buf, size_t n)**
  
  Initializes a mailbox_t object.

- **void chMBReset (mailbox_t *mbp)**

  Resets a mailbox_t object.

- **void 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 a high priority message into a mailbox.

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

  Posts a high priority message into a mailbox.

- **msg_t chMBPostAheadI (mailbox_t *mbp, msg_t msg)**

  Posts a high priority message into a mailbox.

- **msg_t chMBFetchTimeout (mailbox_t *mbp, msg_t *msgp, sysinterval_t timeout)**

  Retrieves a message from a mailbox.

- **msg_t chMBFetchTimeoutS (mailbox_t *mbp, msg_t *msgp, sysinterval_t timeout)**

  Retrieves a message from a mailbox.

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

  Retrieves a message from a mailbox.

- **static size_t chMBGetSizeI (const mailbox_t *mbp)**

  Returns the mailbox buffer size as number of messages.

- **static size_t chMBGetUsedCountI (const mailbox_t *mbp)**

  Returns the number of used message slots into a mailbox.

- **static size_t chMBGetFreeCountI (const mailbox_t *mbp)**

  Returns the number of free message slots into a mailbox.

- **static msg_t chMBPeekI (const mailbox_t *mbp)**

  Returns the next message in the queue without removing it.

- **static void chMBResumeX (mailbox_t *mbp)**

  Terminates the reset state.

### 6.41.2 Macro Definition Documentation
6.41.2.1 __MAILBOX_DATA

#define __MAILBOX_DATA(
    name,
    buffer,
    size )

Value:
{
    (msg_t *)(buffer),
    (msg_t *)(buffer) + size,
    (msg_t *)(buffer),
    (msg_t *)(buffer),
    (size_t)0,
    false,
    __THREADS_QUEUE_DATA(name.qw),
    __THREADS_QUEUE_DATA(name.qr),
}

Data part of a static mailbox initializer.

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

Parameters

<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>

6.41.2.2 MAILBOX_DECL

#define MAILBOX_DECL(
    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>

6.41.3 Function Documentation
6.41.3.1 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>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>out</code></td>
<td><code>mbp</code> the pointer to the <code>mailbox_t</code> structure to be initialized</td>
</tr>
<tr>
<td><code>in</code></td>
<td><code>buf</code> pointer to the messages buffer as an array of <code>msg_t</code></td>
</tr>
<tr>
<td><code>in</code></td>
<td><code>n</code> 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
```

6.41.3.2 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>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>in</code></td>
<td><code>mbp</code> 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:

```
6.41.3.3 chMBResetI()

void chMBResetI ( mailbox_t * mbp )

Resets a mailbox_t object.

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

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

Parameters
- in 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:

![Call Graph Image]

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

**Returns**

The operation status.

**Return values**

<table>
<thead>
<tr>
<th>Return Code</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:

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

Here is the call graph for this function:

```
chMBPostTimeoutS
chMBPostTimeout
chSysLock
chSysUnlock
chDbgCheckS
chMBGetFreeCountI
chSchRescheduleS
chThdDequeueNextI
chThdEnqueueTimeoutS
__dbg_check_lock
__stats_start_measure
_crit_thd
__dbg_check_unlock
__stats_stop_measure
_crit_thd
```

6.41.3.5 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>
</tbody>
</table>

- `TIME_IMMEDIATE` immediate timeout.
- `TIME_INFINITE` no timeout.
Returns

The operation status.

Return values

<table>
<thead>
<tr>
<th>Return value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSG_OK</td>
<td>If 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:

---

### 6.41.3.6 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 <code>mailbox_t</code> 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

| MSG_OK     | if a message has been correctly posted. |
| MSG_RESET  | if the mailbox has been reset.         |
| MSG_TIMEOUT| if the mailbox is full and the message cannot be posted. |

Function Class:

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

Here is the call graph for this function:

6.41.3.7 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><strong>in</strong> mbp</td>
<td>the pointer to an initialized mailbox_t object</td>
</tr>
<tr>
<td><strong>in</strong> msg</td>
<td>the message to be posted on the mailbox</td>
</tr>
<tr>
<td><strong>in</strong> timeout</td>
<td>the number of ticks before the operation timeouts, the following special values are allowed:</td>
</tr>
<tr>
<td></td>
<td>• TIME_IMMEDIATE immediate timeout.</td>
</tr>
<tr>
<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:

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

ChibiOS
Returns

The operation status.

Return values

<table>
<thead>
<tr>
<th>Code</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](call_graph.png)

### 6.41.3.9 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>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>
</tbody>
</table>
Returns

The operation status.

Return values

<table>
<thead>
<tr>
<th>Code</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)

### 6.41.3.10 chMBFetchTimeout()

```c
msg_t chMBFetchTimeout (
    mailbox_t * mbp,
    msg_t * msgp,
    sysinterval_t timeout )
```

Retrieves a message from a mailbox.

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

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mbp</td>
<td>the pointer to an initialized <code>mailbox_t</code> object</td>
</tr>
<tr>
<td>msgp</td>
<td>pointer to a message variable for the received message</td>
</tr>
<tr>
<td>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

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

Function Class:

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

Here is the call graph for this function:

6.41.3.11 chMBFetchTimeoutS()

```c
msg_t chMBFetchTimeoutS ( 
    mailbox_t * mbp,  
    msg_t * msgp,    
    sysinterval_t timeout )
```

Retrieves a message from a mailbox.

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

Parameters

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

Returns

The operation status.

Return values

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

Function Class:

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

Here is the call graph for this function:

6.41.3.12 chMBFetchI()

msg_t chMBFetchI ( 
    mailbox_t * mbp, 
    msg_t * msgp )

Retrieves a message from a mailbox.

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

| in | mbp | the pointer to an initialized mailbox_t object |
| out | msgp | pointer to a message variable for the received message |

Returns

The operation status.

Return values

| 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:

```
6.41.3.13 chMBGetSizeI()

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

Returns the mailbox buffer size as number of messages.

Parameters

| in | mbp | the pointer to an initialized mailbox_t object |
```
6.41 Mailboxes

Returns

The size of the mailbox.

Function Class:

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

6.41.3.14 chMBGetUsedCountI()

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

Returns the number of used message slots into a mailbox.

Parameters

in 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

6.41.3.15 chMBGetFreeCountI()

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

Returns the number of free message slots into a mailbox.
Parameters

| in  | 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]

6.41.3.16 chMBPeek()

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

Returns the next message in the queue without removing it.

Precondition

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

Parameters

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

Returns

The next message in queue.
Function Class:

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

Here is the call graph for this function:

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

### 6.41.3.17 chMBResumeX()

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

Terminates the reset state.

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>in</code></td>
<td><code>mbp</code></td>
<td>the pointer to an initialized <code>mailbox_t</code> object</td>
</tr>
</tbody>
</table>

Function Class:

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

6.42.1 Detailed Description

Macros

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

Data Structures

- `struct pipe_t`
  Structure representing a pipe object.

Functions

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

6.42.2 Macro Definition Documentation
### 6.42.2.1 __PIPE_DATA

```c
#define __PIPE_DATA(
    name,
    buffer,
    size 
)
```

**Value:**

```c

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

### 6.42.2.2 PIPE_DECL

```c
#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>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>name</code></td>
<td>the name of the pipe variable</td>
</tr>
<tr>
<td><code>buffer</code></td>
<td>pointer to the pipe buffer array of <code>uint8_t</code></td>
</tr>
<tr>
<td><code>size</code></td>
<td>number of <code>uint8_t</code> elements in the buffer array</td>
</tr>
</tbody>
</table>

### 6.42.3 Function Documentation

ChibiOS/RT
6.42.3.1 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**

| in  | pp | the pointer to an initialized pipe_t object |
|-----|====|---------------------------------------------|
| in  | bp | pointer to the data buffer                 |
| in  | n  | the maximum amount of data to be transferred, the value 0 is reserved |

**Returns**

The number of bytes effectively transferred.

**Function Class:**

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

6.42.3.2 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**

| in  | pp | the pointer to an initialized pipe_t object   |
|-----|====|-----------------------------------------------|
| out | bp | pointer to the data buffer                    |
| in  | n  | the maximum amount of data to be transferred, the value 0 is reserved |

**Returns**

The number of bytes effectively transferred.
Function Class:

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

### 6.42.3.3 chPipeObjectInit()

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

Initializes a `mailbox_t` object.

**Parameters**

<table>
<thead>
<tr>
<th>in</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.

### 6.42.3.4 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.
6.42.3.5 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>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>pp</td>
<td>the pointer to an initialized pipe_t object</td>
</tr>
<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>• TIME_IMMEDIATE immediate timeout.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• TIME_INFINITE 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.

6.42.3.6 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>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>pp</td>
<td>the pointer to an initialized pipe_t object</td>
</tr>
<tr>
<td>out</td>
<td>bp</td>
<td>the number of bytes to be read, the value 0 is reserved</td>
</tr>
</tbody>
</table>
6.42 Pipes

Returns

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

Function Class:

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

6.42.3.7 chPipeGetSize()

\[
\text{static size_t chPipeGetSize (}
\text{\hspace{1cm}} \text{const pipe_t * pp}) \text{[inline], [static]}
\]

Returns the pipe buffer size as number of bytes.

Parameters

\[
\text{in} \hspace{1cm} \text{pp} \hspace{1cm} \text{the pointer to an initialized pipe_t object}
\]

Returns

The size of the pipe.

Function Class:

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

6.42.3.8 chPipeGetUsedCount()

\[
\text{static size_t chPipeGetUsedCount (}
\text{\hspace{1cm}} \text{const pipe_t * pp}) \text{[inline], [static]}
\]

Returns the number of used byte slots into a pipe.

Parameters

\[
\text{in} \hspace{1cm} \text{pp} \hspace{1cm} \text{the pointer to an initialized pipe_t object}
\]

Returns

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

6.42.3.9 chPipeGetFreeCount()

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

Returns the number of free byte slots into a pipe.

Parameters

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

Returns

The number of empty byte slots.

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

Here is the call graph for this function:

```
chPipeGetFreeCount
  ↓
  chPipeGetSize
  ↓
  chPipeGetUsedCount
```

6.42.3.10 chPipeResume()

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

Terminates the reset state.
Parameters

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

Function Class:

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

6.43.1 Detailed Description

Typedefs

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

Functions

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

6.43.2 Typedef Documentation

6.43.2.1 delegate_veneer_t

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

6.43.2.2 delegate_fn0_t

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

6.43.2.3 delegate_fn1_t

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

6.43.2.4 delegate_fn2_t

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

6.43.2.5 delegate_fn3_t

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

6.43.2.6 delegate_fn4_t

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

6.43.3 Function Documentation

6.43.3.1 __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.

6.43.3.2 __ch_delegate_fn1()

```c
msg_t __ch_delegate_fn1 {
  va_list * argsp 
}
```

Veneer for functions with one parameter.

Parameters

| in | argsp | the list of arguments |

Returns

The function return value.

6.43.3.3 __ch_delegate_fn2()

```c
msg_t __ch_delegate_fn2 {
  va_list * argsp 
}
```

Veneer for functions with two parameters.

Parameters

| in | argsp | the list of arguments |

Returns

The function return value.

6.43.3.4 __ch_delegate_fn3()

```c
msg_t __ch_delegate_fn3 {
  va_list * argsp 
}
```
Veneer for functions with three parameters.
Parameters

| in | argsp | the list of arguments |

Returns

The function return value.

6.43.3.5 __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.

6.43.3.6 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)

6.43.3.7 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)
6.43.3.8  chDelegateDispatchTimeout()

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

Call messages dispatching with timeout.

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

Parameters

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

Returns

The function outcome.

Return values

| MSG_OK | if a function has been called. |
| MSG_TIMEOUT | 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:
6.43.3.9  chDelegateCallDirect0()

static msg_t chDelegateCallDirect0 {
    thread_t * tp,
    delegate_fn0_t func } [inline], [static]

Direct call to a function with no parameters.

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

Parameters

| in | 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:

```
<table>
<thead>
<tr>
<th>chDelegateCallDirect0</th>
<th>__ch_delegate_fn0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>chDelegateCallVeneer</td>
</tr>
<tr>
<td></td>
<td>chMsgSend</td>
</tr>
</tbody>
</table>
```

6.43.3.10  chDelegateCallDirect1()

static msg_t chDelegateCallDirect1 {
    thread_t * tp,
    delegate_fn1_t func,
    msg_t pl } [inline], [static]

Direct call to a function with one parameter.

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

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

Returns

The function return value as a msg_t.

Here is the call graph for this function:

```
chDelegateCallDirect1
__ch_delegate_fn1
chDelegateCallVeneer chMsgSend
```

6.43.3.11  chDelegateCallDirect2()

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

Direct call to a function with two parameters.

Note

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

Parameters

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

The function return value as a msg_t.

Here is the call graph for this function:

6.43.3.12 chDelegateCallDirect3()

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

Direct call to a function with three parameters.

Note

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

Parameters

| in | tp | pointer to the delegate thread |
| in | func | pointer to the function to be called |
| in | p1 | parameter 1 passed as a msg_t |
| in | p2 | parameter 2 passed as a msg_t |
| in | p3 | parameter 3 passed as a msg_t |
Returns

The function return value as a `msg_t`.

Here is the call graph for this function:

![Call Graph](image)

### 6.43.3.13 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>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td><code>tp</code></td>
</tr>
<tr>
<td>in</td>
<td><code>func</code></td>
</tr>
<tr>
<td>in</td>
<td><code>p1</code></td>
</tr>
<tr>
<td>in</td>
<td><code>p2</code></td>
</tr>
<tr>
<td>in</td>
<td><code>p3</code></td>
</tr>
<tr>
<td>in</td>
<td><code>p4</code></td>
</tr>
</tbody>
</table>
Returns

The function return value as a `msg_t`.

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

6.44.1 Detailed Description

Macros

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

Typedefs

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

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

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

Data Structures

- `struct ch_jobs_queue`
  
  Type of a jobs queue.

- `struct ch_job_descriptor`
  
  Type of a job descriptor.

Functions

- `static void chJobObjectInit (jobs_queue_t *jqp, size_t jobsn, job_descriptor_t *jobsbuf, msg_t *msgbuf)`
  
  Initializes a jobs queue object.

- `static job_descriptor_t * chJobGet (jobs_queue_t *jqp)`
  
  Allocates a free job object.

- `static job_descriptor_t * chJobGetI (jobs_queue_t *jqp)`
  
  Allocates a free job object.

- `static job_descriptor_t * chJobGetTimeoutS (jobs_queue_t *jqp, sysinterval_t timeout)`
  
  Allocates a free job object.

- `static job_descriptor_t * chJobGetTimeout (jobs_queue_t *jqp, sysinterval_t timeout)`
  
  Allocates a free job object.

- `static void chJobPostI (jobs_queue_t *jqp, job_descriptor_t *jp)`
  
  Posts a job object.

- `static void chJobPostS (jobs_queue_t *jqp, job_descriptor_t *jp)`
  
  Posts a job object.

- `static void chJobPost (jobs_queue_t *jqp, job_descriptor_t *jp)`
  
  Posts a job object.

- `static void chJobPostAheadI (jobs_queue_t *jqp, job_descriptor_t *jp)`
  
  Posts an high priority job object.

- `static void chJobPostAheadS (jobs_queue_t *jqp, job_descriptor_t *jp)`
  
  Posts an high priority job object.

- `static void chJobPostAhead (jobs_queue_t *jqp, job_descriptor_t *jp)`
  
  Posts an high priority job object.

- `static msg_t chJobDispatch (jobs_queue_t *jqp)`
  
  Waits for a job then executes it.

- `static msg_t chJobDispatchTimeout (jobs_queue_t *jqp, sysinterval_t timeout)`
  
  Waits for a job then executes it.
6.44 Jobs Queues

6.44.2 Macro Definition Documentation

6.44.2.1 MSG_JOB_NULL

#define MSG_JOB_NULL ((msg_t)-2)

Dispatcher return code in case of a JOB_NULL has been received.

6.44.3 Typedef Documentation

6.44.3.1 jobs_queue_t

typedef struct ch_jobs_queue jobs_queue_t

Type of a jobs queue.

6.44.3.2 job_function_t

typedef void(* job_function_t) (void *arg)

Type of a job function.

6.44.3.3 job_descriptor_t

typedef struct ch_job_descriptor job_descriptor_t

Type of a job descriptor.

6.44.4 Function Documentation

6.44.4.1 chJobObjectInit()

static void chJobObjectInit ( 
   jobs_queue_t * jqp, 
   size_t jobsn, 
   job_descriptor_t * jobsbuf, 
   msg_t * msgbuf ) [inline], [static]

Initializes a jobs queue object.
**Parameters**

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

**Function Class:**

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

Here is the call graph for this function:

![Call Graph](image)

### 6.44.4.2 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><strong>in</strong> <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.
6.44 Jobs Queues

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

Here is the call graph for this function:

6.44.4.3 chJobGet()

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:

```
chJobGetI  
|     | 
|     | chGuardedPoolAllocI 
|     | 
|     | | 
|     | | chSemFastWaitI 
|     | | 
|     | | chSemGetCounterI 
```

### 6.44.4.4 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**

```plaintext
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:

```
chJobGetTimeoutS --> chGuardedPoolAllocTimeoutS --> chPoolAllocI --> chSemWaitTimeoutS
```

### 6.44.4.5 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>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 pointer to the allocated job object.

**Return values**

| NULL | if a job object is not available within the specified timeout. |
Function Class:

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

Here is the call graph for this function:

![Call Graph](image)

6.44.4.6 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:

```
<table>
<thead>
<tr>
<th>chJobPostI</th>
<th>chMBPostI</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>chDbgCheckClassI</td>
</tr>
<tr>
<td></td>
<td>chMBGetFreeCountI</td>
</tr>
<tr>
<td></td>
<td>chThdDequeueNextI</td>
</tr>
</tbody>
</table>
```

### 6.44.4.7 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>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)

### 6.44.4.8 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><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:

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

Here is the call graph for this function:

```
chJobPost -> chMBPostTimeout -> chMBPostTimeoutS
        |                        |
        v                        v
chSysLock -> chSysUnlock
```

### 6.44.4.9 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>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td><em>jqp</em> pointer to a jobs_queue_t structure</td>
</tr>
<tr>
<td>in</td>
<td><em>jp</em> 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:

```
chJobPostAheadI
chMBPostAheadI
chDbgCheckClassI
chMBGetFreeCountI
chThdDequeueNextI
```

6.44.4.10  **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><strong>in</strong></th>
<th><strong>jqp</strong></th>
<th>pointer to a <strong>jobs_queue_t</strong> structure</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>in</strong></td>
<td><strong>jp</strong></td>
<td>pointer to the job object to be posted</td>
</tr>
</tbody>
</table>
6.44 Jobs Queues

Function Class:

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

Here is the call graph for this function:

6.44.4.11 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>    pointer to a <code>jobs_queue_t</code> structure</td>
</tr>
<tr>
<td>in</td>
<td><code>jp</code>     pointer to the job object to be posted</td>
</tr>
</tbody>
</table>
Function Class:
Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

```
chJobPostAhead chMBPostAheadTimeout
chMBPostAheadTimeoutS
chSysLock
chSysUnlock
```

### 6.44.4.12 chJobDispatch()

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

Waits for a job then executes it.

**Parameters**

| in | jqp | pointer to a jobs_queue_t structure |

**Returns**

The function outcome.

**Return values**

| MSG_OK    | if a job has been executed. |
| MSG_RESET | if the internal mailbox has been reset. |
| MSG_JOB_NULL | if a JOB_NULL has been received. |
Here is the call graph for this function:

```
chJobDispatch
  └ chMBFetchTimeoutS
     └ chMBFetchTimeout
        └ chGuardedPoolFreeI
           └ chGuardedPoolFree
              └ chSysUnlock
                 └ chSysLock
                    └ chJobDispatch
```

### 6.44.4.13  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**

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSG_OK</td>
<td>if a job has been executed.</td>
</tr>
<tr>
<td>MSG_TIMEOUT</td>
<td>if a timeout occurred.</td>
</tr>
<tr>
<td>MSG_RESET</td>
<td>if the internal mailbox has been reset.</td>
</tr>
<tr>
<td>MSG_JOB_NULL</td>
<td>if a JOB_NULL has been received.</td>
</tr>
</tbody>
</table>
Here is the call graph for this function:
6.45 Memory Management

6.45.1 Detailed Description

Memory Management services.

Modules

- Core Memory Manager
- Memory Heaps
- Memory Pools
6.46 Core Memory Manager

6.46.1 Detailed Description

Core Memory Manager related APIs and services.

Operation mode

The core memory manager is a simplified allocator that only allows to allocate memory blocks without the possibility to free them.
This allocator is meant as a memory blocks provider for the other allocators such as:

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

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

Precondition

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

Note

Compatible with RT and NIL.

Macros

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

Typedefs

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

Data Structures

- struct memcore_t
  Type of memory core object.
Functions

- **void __core_init (void)**
  
  _Low level memory manager initialization._

- **void * chCoreAllocFromBaseI (size_t size, unsigned align, size_t offset)**
  
  _Allocates a memory block starting from the lowest address upward._

- **void * chCoreAllocFromTopI (size_t size, unsigned align, size_t offset)**
  
  _Allocates a memory block starting from the top address downward._

- **void * chCoreAllocFromBase (size_t size, unsigned align, size_t offset)**
  
  _Allocates a memory block starting from the lowest address upward._

- **void * chCoreAllocFromTop (size_t size, unsigned align, size_t offset)**
  
  _Allocates a memory block starting from the top address downward._

- **size_t chCoreGetStatusX (void)**
  
  _Core memory status._

- **static void * chCoreAllocAlignedI (size_t size, unsigned align)**
  
  _Allocates a memory block._

- **static void * chCoreAllocAligned (size_t size, unsigned align)**
  
  _Allocates a memory block._

- **static void * chCoreAlloc (size_t size)**
  
  _Allocates a memory block._

- **static void * chCoreAlloc (size_t size)**
  
  _Allocates a memory block._

Variables

- **memcore_t ch_memcore**
  
  _Memory core descriptor._

6.46.2 Macro Definition Documentation

6.46.2.1 CH_CFG_MEMCORE_SIZE

```c
#define CH_CFG_MEMCORE_SIZE 0
```

Managed RAM size.

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

**Note**

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

**Requires** CH_CFG_USE_MEMCORE.
6.46.2.2 chCoreAllocAlignedWithOffsetI

#define chCoreAllocAlignedWithOffsetI chCoreAllocFromTopI

Allocates a memory block.

Note
This is a generic form with unspecified allocation position.

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

6.46.2.3 chCoreAllocAlignedWithOffset

#define chCoreAllocAlignedWithOffset chCoreAllocFromTop

Allocates a memory block.

Note
This is a generic form with unspecified allocation position.

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

6.46.3 Typedef Documentation

6.46.3.1 memgetfunc_t

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

Memory get function.

6.46.3.2 memgetfunc2_t

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

Enhanced memory get function.
6.46.4 Function Documentation

6.46.4.1 __core_init()

```c
void __core_init (
    void )
```

Low level memory manager initialization.

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

6.46.4.2 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 `offset + size` bytes. The returned pointer has `offset` bytes before its address and `size` bytes after.

Parameters

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

Returns

A pointer to the allocated memory block.

Return values

- `NULL`: allocation failed, core memory exhausted.

Function Class:

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

![Call Graph](image)

### 6.46.4.3 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>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td><code>size</code> the size of the block to be allocated.</td>
</tr>
<tr>
<td>in</td>
<td><code>align</code> desired memory alignment</td>
</tr>
<tr>
<td>in</td>
<td><code>offset</code> 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
```

6.46.4.4 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](image.png)

6.46.4.5  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:

6.46.4.6 chCoreGetStatusX()

size_t chCoreGetStatusX ( void )

Core memory status.

Returns

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

Function Class:

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

6.46.4.7 chCoreAllocAlignedI()

static void* chCoreAllocAlignedI ( size_t size, unsigned align ) [inline], [static]

Allocates a memory block.

The allocated block is guaranteed to be properly aligned to the specified alignment.

Note

This is a generic form with unspecified allocation position.
Parameters

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

Returns

A pointer to the allocated memory block.

Return values

- **NULL**: allocation failed, core memory exhausted.

Function Class:

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

### 6.46.4.8 chCoreAllocAligned()

`static void* chCoreAllocAligned (size_t size, unsigned align) [inline], [static]`

Allocates a memory block.

The allocated block is guaranteed to be properly aligned to the specified alignment.

Note

This is a generic form with unspecified allocation position.

Parameters

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

Returns

A pointer to the allocated memory block.

Return values

- **NULL**: allocation failed, core memory exhausted.
6.46 Core Memory Manager

Function Class:

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

6.46.4.9 chCoreAllocI()

static void* chCoreAllocI ( 
      size_t size ) [inline], [static]

Allocates a memory block.

The allocated block is guaranteed to be properly aligned for a pointer data type.

Note

This is a generic form with unspecified allocation position.

Parameters

| in  | size | the size of the block to be allocated |

Returns

A pointer to the allocated memory block.

Return values

| NULL | allocation failed, core memory exhausted |

Function Class:

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

6.46.4.10 chCoreAlloc()

static void* chCoreAlloc ( 
      size_t size ) [inline], [static]

Allocates a memory block.

The allocated block is guaranteed to be properly aligned for a pointer data type.

Note

This is a generic form with unspecified allocation position.
Parameters

| in  | size | the size of the block to be allocated. |

Returns

A pointer to the allocated memory block.

Return values

| NULL | allocation failed, core memory exhausted. |

Function Class:

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

6.46.5 Variable Documentation

6.46.5.1 ch_memcore

memcore_t ch_memcore

Memory core descriptor.
6.47 Memory Heaps

6.47.1 Detailed Description

Heap Allocator related APIs.

Operation mode

The heap allocator implements a first-fit strategy and its APIs are functionally equivalent to the usual malloc() and free() library functions. The main difference is that the OS heap APIs are guaranteed to be thread safe and there is the ability to return memory blocks aligned to arbitrary powers of two.

Precondition

In order to use the heap APIs the CH_CFG_USE_HEAP option must be enabled in chconf.h.

Note

Compatible with RT and NIL.

Macros

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

Typedefs

- typedef struct memory_heap memory_heap_t
  
  Type of a memory heap.
- typedef union heap_header heap_header_t
  
  Type of a memory heap header.

Data Structures

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

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

Variables

- `static memory_heap_t default_heap`
  Default heap descriptor.

6.47.2 Macro Definition Documentation

6.47.2.1 CH_HEAP_ALIGNMENT

#define CH_HEAP_ALIGNMENT 8U

Minimum alignment used for heap.

Note

Cannot use the sizeof operator in this macro.

6.47.2.2 CH_HEAP_AREA

#define CH_HEAP_AREA(
    name,
    size
)

Value:

ALIGNED_VAR(CH_HEAP_ALIGNMENT) \t
uint8_t name[MEM_ALIGN_NEXT((size), CH_HEAP_ALIGNMENT)]

Allocation of an aligned static heap buffer.
6.47 Memory Heaps

6.47.3 Typedef Documentation

6.47.3.1 memory_heap_t

typedef struct memory_heap memory_heap_t

Type of a memory heap.

6.47.3.2 heap_header_t

typedef union heap_header heap_header_t

Type of a memory heap header.

6.47.4 Function Documentation

6.47.4.1 __heap_init()

void __heap_init (  
    void )

Initializes the default heap.

Function Class:

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

6.47.4.2 chHeapObjectInit()

void chHeapObjectInit (  
    memory_heap_t * heapp,  
    void * buf,  
    size_t size )

Initializes a memory heap from a static memory area.

Note

The heap buffer base and size are adjusted if the passed buffer is not aligned to CH_HEAP_ALIGNMENT. This mean that the effective heap size can be less than size.
Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>heapp</td>
<td>pointer to the memory heap descriptor to be initialized</td>
</tr>
<tr>
<td>in</td>
<td>buf</td>
<td>heap buffer base</td>
</tr>
<tr>
<td>in</td>
<td>size</td>
<td>heap size</td>
</tr>
</tbody>
</table>

Function Class:

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

### 6.47.4.3 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>Type</th>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>heapp</td>
<td>pointer to a heap descriptor or NULL in order to access the default heap.</td>
</tr>
<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

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NULL</td>
<td>if the block cannot be allocated.</td>
</tr>
</tbody>
</table>

Function Class:

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

### 6.47.4.4 chHeapFree()

```c
void chHeapFree (  
    void * p  
)
```

Frees a previously allocated memory block.
6.47 Memory Heaps

Parameters

\[
\text{\textbf{in} } p \quad \text{pointer to the memory block to be freed}
\]

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

6.47.4.5 \texttt{chHeapStatus()}

\[
\text{size_t chHeapStatus (}
\quad \text{memory_heap_t } \ast \text{ heapp},
\quad \text{size_t } \ast \text{ totalp},
\quad \text{size_t } \ast \text{ largestp})
\]

Reports the heap status.

Note
This function is meant to be used in the test suite, it should not be really useful for the application code.

Parameters

\[
\begin{array}{|c|c|}
\hline
\text{in} & \text{heapp} & \text{pointer to a heap descriptor or NULL in order to access the default heap.} \\
\hline
\text{in} & \text{totalp} & \text{pointer to a variable that will receive the total fragmented free space or NULL} \\
\hline
\text{in} & \text{largestp} & \text{pointer to a variable that will receive the largest free free block found space or NULL} \\
\hline
\end{array}
\]

Returns
The number of fragments in the heap.

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

6.47.4.6 \texttt{chHeapAlloc()}

\[
\text{static void* chHeapAlloc (}
\quad \text{memory_heap_t } \ast \text{ heapp,}
\quad \text{size_t } \text{size } ) \text{ [inline], [static]}
\]

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>heapp</code></td>
<td>pointer to a heap descriptor or <code>NULL</code> in order to access the default heap.</td>
</tr>
<tr>
<td><code>size</code></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>
</tbody>
</table>

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:

```
chHeapAlloc ← chHeapAllocAligned
```

6.47.4.7 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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>p</code></td>
<td>pointer to the memory block</td>
</tr>
</tbody>
</table>
Returns

Size of the block.

Function Class:

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

6.47.5 Variable Documentation

6.47.5.1 default_heap

```c
memory_heap_t default_heap [static]
```

Default heap descriptor.
6.48 Memory Pools

6.48.1 Detailed Description

Memory Pools related APIs and services.

Operation mode

The Memory Pools APIs allow to allocate/free fixed size objects in constant time and reliably without memory fragmentation problems. Memory Pools do not enforce any alignment constraint on the contained object however the objects must be properly aligned to contain a pointer to void.

Precondition

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

Note

Compatible with RT and NIL.

Macros

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

Data Structures

• struct pool_header
  Memory pool free object header.
• struct memory_pool_t
  Memory pool descriptor.
• struct guarded_memory_pool_t
  Guarded memory pool descriptor.
Functions

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

6.48.2 Macro Definition Documentation

ChibiOS/RT
6.48.2.1  __MEMORYPOOL_DATA

#define __MEMORYPOOL_DATA(
    name,
    size,
    align,
    provider ) {NULL, size, align, provider}

Data part of a static memory pool initializer.

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

Parameters

<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>
<tr>
<td>in</td>
<td>provider</td>
<td>memory provider function for the memory pool</td>
</tr>
</tbody>
</table>

6.48.2.2  MEMORYPOOL_DECL

#define MEMORYPOOL_DECL(
    name,
    size,
    align,
    provider
) memory_pool_t name = __MEMORYPOOL_DATA(name, size, align, provider)

Static memory pool initializer.

Statically initialized memory pools require no explicit initialization using chPoolInit().

Parameters

<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>
<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>

6.48.2.3  __GUARDEDMEMORYPOOL_DATA

#define __GUARDEDMEMORYPOOL_DATA(
    name,
    size,
    align )
Value:
{
    __SEMAPHORE_DATA(name.sem, {cnt_t}0),
    __MEMORYPOOL_DATA(NULL, size, align, NULL)
}

Data part of a static guarded memory pool initializer.

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

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>name</th>
<th>the name of the memory pool variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>size</td>
<td>size of the memory pool contained objects</td>
</tr>
<tr>
<td>in</td>
<td>align</td>
<td>required memory alignment</td>
</tr>
</tbody>
</table>

6.48.2.4 GUARDEDMEMORYPOOL_DECL

#define GUARDEDMEMORYPOOL_DECL(name, size, align)

    guarded_memory_pool_t name = __GUARDEDMEMORYPOOL_DATA(name, size, align)

Static guarded memory pool initializer.

Statically initialized guarded memory pools require no explicit initialization using chGuardedPoolInit().

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>name</th>
<th>the name of the guarded memory pool variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>size</td>
<td>size of the memory pool contained objects</td>
</tr>
<tr>
<td>in</td>
<td>align</td>
<td>required memory alignment</td>
</tr>
</tbody>
</table>

6.48.3 Function Documentation

6.48.3.1 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>out</th>
<th>mp</th>
<th>pointer to a memory_pool_t structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>size</td>
<td>the size of the objects contained in this memory pool, the minimum accepted size is the size of a pointer to void.</td>
</tr>
<tr>
<td>in</td>
<td>align</td>
<td>required memory alignment</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.

6.48.3.2 chPoolLoadArray()

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

Loads a memory pool with an array of static objects.

Precondition

The memory pool must already be initialized.
The array elements must be of the right size for the specified memory pool.
The array elements size must be a multiple of the alignment requirement for the pool.

Postcondition

The memory pool contains the elements of the input array.

Parameters

<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>p</td>
<td>pointer to the array first element</td>
</tr>
<tr>
<td>in</td>
<td>n</td>
<td>number of elements in the array</td>
</tr>
</tbody>
</table>
6.48 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:

6.48.3.3 chPoolAllocI()

```c
void * chPoolAllocI ( memory_pool_t * mp )
```

Allocates an object from a memory pool.

Precondition
The memory pool must already be initialized.

Parameters

```
in mp pointer to a memory_pool_t structure
```

Returns
The pointer to the allocated object.

Return values

```
NULL if pool is empty.
```
Function Class:

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

Here is the call graph for this function:

![Call Graph](image)

### 6.48.3.4 chPoolAlloc()

```c
void * chPoolAlloc (    
    memory_pool_t * mp  )
```

Allocates an object from a memory pool.

**Precondition**

The memory pool must already be initialized.

**Parameters**

| in  | mp | pointer to a `memory_pool_t` structure |

**Returns**

The pointer to the allocated object.

**Return values**

- `NULL` if pool is empty.
Function Class:

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

Here is the call graph for this function:

6.48.3.5 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>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:

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
```

6.48.3.6 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:

![Call Graph Diagram]

### 6.48.3.7 chGuardedPoolObjectInitAligned()

```c
void chGuardedPoolObjectInitAligned ( 
    guarded_memory_pool_t * gmp, 
    size_t size, 
    unsigned align )
```

Initializes an empty guarded memory pool.

**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>size</code></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><code>align</code></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:

6.48.3.8 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>in</th>
<th>gmp</th>
<th>pointer to a guarded_memory_pool_t structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>p</td>
<td>pointer to the array first element</td>
</tr>
<tr>
<td>in</td>
<td>n</td>
<td>number of elements in the array</td>
</tr>
</tbody>
</table>
6.48 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:

![](chGuardedPoolLoadArray chGuardedPoolAdd chGuardedPoolFree)

6.48.3.9 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>in</th>
<th>gmp</th>
<th>pointer to a guarded_memory_pool_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 the operation timed out. |
Function Class:

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

Here is the call graph for this function:

![Call Graph](image)

### 6.48.3.10 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)

### 6.48.3.11 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><strong>gmp</strong> pointer to a <code>guarded_memory_pool_t</code> structure</td>
</tr>
<tr>
<td>in</td>
<td><strong>objp</strong> 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:

![Call Graph](image)

6.48.3.12 chPoolObjectInit()

static void chPoolObjectInit {
    memory_pool_t * mp,
    size_t size,
    memgetfunc_t provider ) [inline], [static]

Initializes an empty memory pool.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>out mp</td>
<td>pointer to a memory_pool_t structure</td>
<td></td>
</tr>
<tr>
<td>in 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>
<td></td>
</tr>
<tr>
<td>in provider</td>
<td>memory provider function for the memory pool or NULL if the pool is not allowed to grow automatically</td>
<td></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
```

### 6.48.3.13 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>Type</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:

```
6.48.3.14   chPoolAddI()

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

This function 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>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:

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

```
|   chPoolAddI   | chPoolFree | chDbgCheckClassI |
```

### 6.48.3.15 chGuardedPoolObjectInit()

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

Initializes an empty guarded memory pool.

**Parameters**

<table>
<thead>
<tr>
<th>out</th>
<th>gmp</th>
<th>pointer to a guarded_memory_pool_t structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>size</td>
<td>the size of the objects contained in this guarded memory pool, the minimum accepted size is the size of a pointer to void.</td>
</tr>
</tbody>
</table>

**Function Class:**

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

Here is the call graph for this function:

```
|   chGuardedPoolObjectInit   | chGuardedPoolObjectInitAligned | chPoolObjectInitAligned |
```

### 6.48.3.16 chGuardedPoolGetCounterI()

```c
static cnt_t chGuardedPoolGetCounterI ( 
    guarded_memory_pool_t * gmp ) [inline], [static]
```

Gets the count of objects in a guarded memory pool.
Precondition

The guarded memory pool must be already been initialized.

Parameters

| in | gmp | pointer to a guarded_memory_pool_t structure |

Returns

The counter of the guard semaphore.

Function Class:

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

Here is the call graph for this function:

```
chGuardedPoolGetCounterI chSemGetCounterI chDbgCheckClassI
```

6.48.3.17 chGuardedPoolAllocI()

```
static void* chGuardedPoolAllocI ( 
    guarded_memory_pool_t * gmp ) [inline], [static]
```

Allocates an object from a guarded memory pool.

Precondition

The guarded memory pool must be already been initialized.

Parameters

| in | gmp | pointer to a guarded_memory_pool_t structure |

Returns

The pointer to the allocated object.
6.48 Memory Pools

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:

```
<table>
<thead>
<tr>
<th>chGuardedPoolAllocI</th>
<th>chPoolAllocI</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>chSemFastWaitI</td>
</tr>
<tr>
<td></td>
<td>chSemGetCounterI</td>
</tr>
<tr>
<td></td>
<td>chDbgCheckClassI</td>
</tr>
</tbody>
</table>
```

6.48.3.18 chGuardedPoolFreeI()

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

Releases an object into a guarded memory pool.

Precondition

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

Parameters

| in  | gmp  | pointer to a guarded_memory_pool_t structure |
| in  | objp | the pointer to the object to be released   |

Function Class:

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

6.48.3.19 chGuardedPoolFreeS()

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

Releases an object into a guarded memory pool.

Precondition

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

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>gmp</th>
<th>pointer to a guarded_memory_pool_t structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>objp</td>
<td>the pointer to the object to be released</td>
</tr>
</tbody>
</table>
## 6.48 Memory Pools

**Function Class:**

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

Here is the call graph for this function:

![Call Graph](image)

### 6.48.3.20 `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>in</th>
<th><code>gmp</code></th>
<th>pointer to a <code>guarded_memory_pool_t</code> structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td><code>objp</code></td>
<td>the pointer to the object to be added</td>
</tr>
</tbody>
</table>
Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

```
chGuardedPoolAdd
  |   |
  v   v
chGuardedPoolFree
  |   |
  v   v
chSchRescheduleS
  |   |
  v   v
chSysLock
  |   |
  v   v
chSysUnlock
```

### 6.48.3.21 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>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 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]

### 6.48.3.22 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**

- **in** `gmp` : pointer to a `guarded_memory_pool_t` structure
- **in** `objp` : the pointer to the object to be added
Function Class:

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

Here is the call graph for this function:
6.49 Complex Services

6.49.1 Detailed Description

Modules

- Objects FIFOs
- Objects Caches
- Dynamic Objects Factory
6.50 Objects FIFOs

6.50.1 Detailed Description

Typedefs

- typedef struct ch_objects_fifo objects_fifo_t
  
  Type of an objects FIFO.

Data Structures

- struct ch_objects_fifo
  
  Type of an objects FIFO.

Functions

- static void chFifoObjectInitAligned (objects_fifo_t *ofp, size_t objsize, size_t objn, unsigned objalign, void *objbuf, msg_t *msgbuf)
  
  Initializes a FIFO object.

- static void chFifoObjectInit (objects_fifo_t *ofp, size_t objsize, size_t objn, void *objbuf, msg_t *msgbuf)
  
  Initializes a FIFO object.

- static void chFifoTakeObjectI (objects_fifo_t *ofp)
  
  Allocates a free object.

- static void chFifoTakeObjectTimeoutS (objects_fifo_t *ofp, sysinterval_t timeout)
  
  Allocates a free object.

- static void chFifoTakeObjectTimeout (objects_fifo_t *ofp, sysinterval_t timeout)
  
  Allocates a free object.

- static void chFifoReturnObjectI (objects_fifo_t *ofp, void *objp)
  
  Releases a fetched object.

- static void chFifoReturnObjectS (objects_fifo_t *ofp, void *objp)
  
  Releases a fetched object.

- static void chFifoReturnObject (objects_fifo_t *ofp, void *objp)
  
  Releases a fetched object.

- static void chFifoSendObjectI (objects_fifo_t *ofp, void *objp)
  
  Posts an object.

- static void chFifoSendObjectS (objects_fifo_t *ofp, void *objp)
  
  Posts an object.

- static void chFifoSendObject (objects_fifo_t *ofp, void *objp)
  
  Posts an object.

- static void chFifoSendObjectAheadI (objects_fifo_t *ofp, void *objp)
  
  Posts an high priority object.

- static void chFifoSendObjectAheadS (objects_fifo_t *ofp, void *objp)
  
  Posts an high priority object.

- static void chFifoSendObjectAhead (objects_fifo_t *ofp, void *objp)
  
  Posts an high priority object.

- static msg_t chFifoReceiveObjectI (objects_fifo_t *ofp, void **objpp)
  
  Fetches an object.

- static msg_t chFifoReceiveObjectTimeoutS (objects_fifo_t *ofp, void **objpp, sysinterval_t timeout)
  
  Fetches an object.

- static msg_t chFifoReceiveObjectTimeout (objects_fifo_t *ofp, void **objpp, sysinterval_t timeout)
  
  Fetches an object.
6.50 Objects FIFOs

6.50.2 Typedef Documentation

6.50.2.1 objects_fifo_t

typedef struct ch_objects_fifo objects_fifo_t

Type of an objects FIFO.

6.50.3 Function Documentation

6.50.3.1 chFifoObjectInitAligned()

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

Initializes a FIFO object.

Precondition

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

Parameters

<table>
<thead>
<tr>
<th>Param</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>ofp</td>
<td>pointer to a objects_fifo_t structure</td>
</tr>
<tr>
<td>in</td>
<td>objsize</td>
<td>size of objects</td>
</tr>
<tr>
<td>in</td>
<td>objn</td>
<td>number of objects available</td>
</tr>
<tr>
<td>in</td>
<td>objalign</td>
<td>required objects alignment</td>
</tr>
<tr>
<td>in</td>
<td>objbuf</td>
<td>pointer to the buffer of objects, it must be able to hold objn objects of objsize size with objalign alignment</td>
</tr>
<tr>
<td>in</td>
<td>msgbuf</td>
<td>pointer to the buffer of messages, it must be able to hold objn messages</td>
</tr>
</tbody>
</table>
Function Class:

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

Here is the call graph for this function:

![Call Graph]

6.50.3.2 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>out ofp</td>
<td>pointer to a objects_fifo_t structure</td>
</tr>
<tr>
<td>in objsize</td>
<td>size of objects</td>
</tr>
<tr>
<td>in objn</td>
<td>number of objects available</td>
</tr>
<tr>
<td>in objbuf</td>
<td>pointer to the buffer of objects, it must be able to hold objn objects of objsize size</td>
</tr>
<tr>
<td>in msgbuf</td>
<td>pointer to the buffer of messages, it must be able to hold objn messages</td>
</tr>
</tbody>
</table>
Function Class:
Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

Here is the call graph for this function:

```
chFifoObjectInit
chFifoObjectInitAligned
chGuardedPoolLoadArray
chGuardedPoolObjectInitAligned
chMBObjectInit
```

### 6.50.3.3 chFifoTakeObjectI()

```c
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
```

### 6.50.3.4 chFifoTakeObjectTimeoutS()

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 <code>objects_fifo_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 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.
6.50 Objects FIFOs

Here is the call graph for this function:

```
chFifoTakeObjectTimeout() -> chSemaphoreWaitTimeout() -> chGuardedPoolAllocTimeout() -> chPoolAlloc()
```

6.50.3.5 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:

![Call Graph]

6.50.3.6 `chFifoReturnObjectI()`

```c
static void chFifoReturnObjectI ( objects_fifo_t * ofp,
                                  void * objp ) [inline], [static]
```

Releases a fetched object.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th><code>ofp</code></th>
<th>pointer to a <code>objects_fifo_t</code> structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td><code>objp</code></td>
<td>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]
### 6.50.3.7 chFifoReturnObjectS()

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

Releases a fetched object.

**Parameters**

- **in** `ofp` pointer to a `objects_fifo_t` structure
- **in** `objp` pointer to the object to be released

**Function Class:**

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

Here is the call graph for this function:

![Call Graph](image.png)

### 6.50.3.8 chFifoReturnObject()

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

Releases a fetched object.

**Parameters**

- **in** `ofp` pointer to a `objects_fifo_t` structure
- **in** `objp` pointer to the object to be released
Function Class:

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

Here is the call graph for this function:

![Call Graph Image]

6.50.3.9 chFifoSendObjectI()

```c
static void chFifoSendObjectI ( 
    objects_fifo_t ∗ ofp,
    void ∗ objp ) [inline], [static]
```

Posts an object.

**Note**

By design the object can be always immediately posted.

**Parameters**

| in |  ofp  | pointer to a objects_fifo_t structure |
| in |  objp  | pointer to the object to be posted |
Function Class:

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

Here is the call graph for this function:

![Call Graph]

6.50.3.10 chFifoSendObjectS()

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

Posts an object.

Note

By design the object can be always immediately posted.

Parameters

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></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 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:

```
chFifoSendObjectS       chMBPostTimeoutS       chDbgCheckClassS
                        chMBGetFreeCountI       chSchRescheduleS
                        chThdDequeueNextI
                        chThdEnqueueTimeoutS
```

### 6.50.3.11 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>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td><code>ofp</code> pointer to a <code>objects_fifo_t</code> structure</td>
</tr>
<tr>
<td>in</td>
<td><code>objp</code> pointer to the object to be released</td>
</tr>
</tbody>
</table>
Function Class:

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

Here is the call graph for this function:

![Call Graph]

### 6.50.3.12 chFifoSendObjectAheadI()

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

Posts an high priority object.

Note

By design the object can be always immediately posted.

Parameters

<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:

```
chFifoSendObjectAheadI  chMBPostAheadI
   chMBCntFreeCountI
   chThdDequeueNextI
```

6.50.3.13  **chFifoSendObjectAheadS()**

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

Posts an high priority object.

**Note**

By design the object can be always immediately posted.

**Parameters**

<table>
<thead>
<tr>
<th>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 **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:

```
chFifoSendObjectAheadS  chMBPostAheadTimeoutS
   chMBCntFreeCountI
   chSchSetSchedulerS
   chSchRescheduleS
   chThdDequeueNextI
   chThdEnqueueTimeoutS
```
6.50.3.14 chFifoSendObjectAhead()

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

Posts an high priority object.

Note

By design the object can be always immediately posted.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>ofp</th>
<th>pointer to a objects_fifo_t structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>objp</td>
<td>pointer to the object to be released</td>
</tr>
</tbody>
</table>

Function Class:

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

Here is the call graph for this function:

![Call Graph]

6.50.3.15 chFifoReceiveObjectI()

static msg_t chFifoReceiveObjectI (  
  objects_fifo_t * ofp,  
  void ** objpp ) [inline], [static]

Fetches an object.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>ofp</th>
<th>pointer to a objects_fifo_t structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>objpp</td>
<td>pointer to the fetched object reference</td>
</tr>
</tbody>
</table>
Returns

The operation status.

Return values

<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:

![Call Graph](image)

6.50.3.16 chFifoReceiveObjectTimeoutS()

```c
static msg_t chFifoReceiveObjectTimeoutS ( 
    objects_fifo_t * ofp,
    void ** objpp,
    sysinterval_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_INFINITE 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:

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:

```
chFifoReceiveObjectTimeoutS  
chMBFetchTimeoutS  
chDbgCheckClassS  
chMBGetUsedCountI  
chSchRescheduleS  
chThdDequeueNextI  
chThdEnqueueTimeoutS
```

### 6.50.3.17 chFifoReceiveObjectTimeout()

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

Fetches an object.

**Parameters**

<table>
<thead>
<tr>
<th></th>
<th>in</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ofp</strong></td>
<td>pointer to a objects_fifo_t structure</td>
<td></td>
</tr>
<tr>
<td><strong>objpp</strong></td>
<td>pointer to the fetched object reference</td>
<td></td>
</tr>
<tr>
<td><strong>timeout</strong></td>
<td>the number of ticks before the operation timeouts, the following special values are allowed:</td>
<td></td>
</tr>
</tbody>
</table>

- **TIME_IMMEDIATE** immediate timeout.
- **TIME_INFINITE** no timeout.

**Returns**

The operation status.
Return values

<table>
<thead>
<tr>
<th>MSG_OK</th>
<th>if an object has been correctly fetched.</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSG_TIMEOUT</td>
<td>if the operation has timed out.</td>
</tr>
</tbody>
</table>

Function Class:

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

Here is the call graph for this function:
6.51 Objects Caches

6.51.1 Detailed Description

Cached objects flags

- #define OC_FLAG_INLRU 0x00000001U
- #define OC_FLAG_INHASH 0x00000002U
- #define OC_FLAG_SHARED 0x00000004U
- #define OC_FLAG_NOTSYNC 0x00000008U
- #define OC_FLAG_LAZYWRITE 0x00000010U
- #define OC_FLAG_FORGET 0x00000020U

Typedefs

- typedef uint32_t oc_flags_t
  
  Flags of cached objects.
- typedef struct ch_oc_hash_header oc_hash_header_t
  
  Type of an hash element header.
- typedef struct ch_oc_lru_header oc_lru_header_t
  
  Type of an LRU element header.
- typedef struct ch_oc_object oc_object_t
  
  Type of a cached object.
- typedef struct ch_objects_cache objects_cache_t
  
  Type of a cache object.
- typedef bool(oc_readf_t) (objects_cache_t *ocp, oc_object_t *objp, bool async)
  
  Object read function.
- typedef bool(oc_writef_t) (objects_cache_t *ocp, oc_object_t *objp, bool async)
  
  Object write function.

Data Structures

- struct ch_oc_hash_header
  
  Structure representing an hash table element.
- struct ch_oc_lru_header
  
  Structure representing an hash table element.
- struct ch_oc_object
  
  Structure representing a cached object.
- struct ch_objects_cache
  
  Structure representing a cache object.
Functions

- static oc_object_t * hash_get_s (objects_cache_t *ocp, uint32_t group, uint32_t key)
  Returns an object pointer from the cache, if present.
- static oc_object_t * lru_get_last_s (objects_cache_t *ocp)
  Gets the least recently used object buffer from the LRU list.
- void chCacheObjectInit (objects_cache_t *ocp, ucnt_t hashn, oc_hash_header_t *hashp, ucnt_t objn, size_t objsz, void *objvp, oc_readf_t readf, oc_writef_t writef)
  Initializes a objects_cache_t object.
- oc_object_t * chCacheGetObject (objects_cache_t *ocp, uint32_t group, uint32_t key)
  Retrieves an object from the cache.
- void chCacheReleaseObject (objects_cache_t *ocp, oc_object_t *objp)
  Releases an object into the cache.
- bool chCacheReadObject (objects_cache_t *ocp, oc_object_t *objp, bool async)
  Reads object data from the storage.
- bool chCacheWriteObject (objects_cache_t *ocp, oc_object_t *objp, bool async)
  Writes the object data back to storage.
- static void chCacheReleaseObject (objects_cache_t *ocp, oc_object_t *objp)
  Releases an object into the cache.

6.51.2 Typedef Documentation

6.51.2.1 oc_flags_t

typedef uint32_t oc_flags_t

Flags of cached objects.

6.51.2.2 oc_hash_header_t

typedef struct ch_oc_hash_header oc_hash_header_t

Type of an hash element header.

6.51.2.3 oc_lru_header_t

typedef struct ch_oc_lru_header oc_lru_header_t

Type of an LRU element header.
6.51.2.4 oc_object_t

typedef struct ch_oc_object oc_object_t

Type of a cached object.

6.51.2.5 objects_cache_t

typedef struct ch_objects_cache objects_cache_t

Type of a cache object.

6.51.2.6 oc_readf_t

typedef bool(* oc_readf_t) (objects_cache_t *ocp, oc_object_t *objp, bool async)

Object read function.

<table>
<thead>
<tr>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>in</strong> ocp</td>
</tr>
<tr>
<td><strong>in</strong> async</td>
</tr>
</tbody>
</table>

6.51.2.7 oc_writef_t

typedef bool(* oc_writef_t) (objects_cache_t *ocp, oc_object_t *objp, bool async)

Object write function.

<table>
<thead>
<tr>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>in</strong> ocp</td>
</tr>
<tr>
<td><strong>in</strong> async</td>
</tr>
</tbody>
</table>

6.51.3 Function Documentation
### 6.51.3.1 hash_get_s()

```c
static oc_object_t * hash_get_s ( objects_cache_t * ocp,
             uint32_t group,
             uint32_t key ) [static]
```

Returns an object pointer from the cache, if present.

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>Input</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>out</code></td>
<td><code>ocp</code></td>
<td>pointer to the <code>objects_cache_t</code> structure to be updated</td>
</tr>
<tr>
<td><code>in</code></td>
<td><code>group</code></td>
<td>object group identifier</td>
</tr>
<tr>
<td><code>in</code></td>
<td><code>key</code></td>
<td>object identifier within the group initialized</td>
</tr>
</tbody>
</table>

**Returns**

The pointer to the retrieved object.

**Return values**

- `NULL` if the object is not in cache.

**Function Class:**

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

### 6.51.3.2 lru_get_last_s()

```c
static oc_object_t * lru_get_last_s ( objects_cache_t * ocp ) [static]
```

Gets the least recently used object buffer from the LRU list.

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>Input</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>out</code></td>
<td><code>ocp</code></td>
<td>pointer to the <code>objects_cache_t</code> structure to be updated</td>
</tr>
</tbody>
</table>

**Returns**

The pointer to the retrieved object.

**Function Class:**

Not an API, this function is for internal use only.
6.51.3.3 chCacheObjectInit()

void chCacheObjectInit {
    objects_cache_t * ocp,
    ucnt_t hashn,
    oc_hash_header_t * hashp,
    ucnt_t objn,
    size_t objsz,
    void * objvp,
    oc_readf_t readf,
    oc_writef_t writef }

Initializes a objects_cache_t object.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>ocp</td>
</tr>
<tr>
<td>in</td>
<td>hashn</td>
</tr>
<tr>
<td>in</td>
<td>hashp</td>
</tr>
<tr>
<td>in</td>
<td>objn</td>
</tr>
<tr>
<td>in</td>
<td>objsz</td>
</tr>
<tr>
<td>in</td>
<td>objvp</td>
</tr>
<tr>
<td>in</td>
<td>readf</td>
</tr>
<tr>
<td>in</td>
<td>writef</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:

```
chCacheObjectInit -> chSemObjectInit -> ch_queue_init
```

6.51.3.4 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:

6.51.3.5 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

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>in</strong></td>
<td><strong>ocp</strong></td>
<td>pointer to the objects_cache_t structure</td>
</tr>
<tr>
<td><strong>in</strong></td>
<td><strong>objp</strong></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.
6.51.3.6 chCacheReadObject()

bool chCacheReadObject (  
    objects_cache_t * ocp,  
    oc_object_t * objp,  
    bool async )

Reads object data from the storage.

Note
In case of asynchronous operation an error condition is not reported by this function.

Parameters

<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>
<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

| false | if the operation succeeded. |
| true  | if the synchronous read operation failed. |

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

6.51.3.7 chCacheWriteObject()

bool chCacheWriteObject (  
    objects_cache_t * ocp,  
    oc_object_t * objp,  
    bool async )

Writes the object data back to storage.

Note
In case of asynchronous operation an error condition is not reported by this function.
Parameters

<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>
<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

<table>
<thead>
<tr>
<th>false</th>
<th>if the operation succeeded.</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</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.

6.51.3.8 chCacheReleaseObject()

static void chCacheReleaseObject (  
    objects_cache_t * ocp,  
    oc_object_t * objp ) [inline], [static]

Releases an object into the cache.

Note

This function gives a meaning to the following flags:

- OC_FLAG_INLRU must be cleared.
- OC_FLAG_INHASH must be set.
- OC_FLAG_SHARED must be cleared.
- OC_FLAG_NOTSYNC invalidates the object and queues it on the LRU tail.
- OC_FLAG_LAZYWRITE is ignored and kept, a write will occur when the object is removed from the LRU list (lazy write).
Function Class:

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

Here is the call graph for this function:
6.52 Dynamic Objects Factory

6.52.1 Detailed Description

The object factory is a subsystem that allows to:

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

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

Precondition

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

Note

Compatible with RT and NIL.

Macros

- #define CH_CFG_FACTORY_MAX_NAMES_LENGTH 8
  Maximum length for object names.
- #define CH_CFG_FACTORY_OBJECTS_REGISTRY TRUE
  Enables the registry of generic objects.
- #define CH_CFG_FACTORY_GENERIC_BUFFERS TRUE
  Enables factory for generic buffers.
- #define CH_CFG_FACTORY_SEMAPHORES TRUE
  Enables factory for semaphores.
- #define CH_CFG_FACTORY_SEMAPHORES FALSE
  Enables factory for semaphores.
- #define CH_CFG_FACTORY_MAILBOXES TRUE
  Enables factory for mailboxes.
- #define CH_CFG_FACTORY_MAILBOXES FALSE
  Enables factory for mailboxes.
- #define CH_CFG_FACTORY_OBJ_FIFOS TRUE
  Enables factory for objects FIFOs.
- #define CH_CFG_FACTORY_OBJ_FIFOS 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_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_objects_fifo
  Type of a dynamic buffer object.
- struct ch_dyn_objects_fifo
  Type of a dynamic buffer object.
- struct ch_objects_factory
  Type of the factory main object.
Functions

- `void __factory_init (void)`
  Initializes the objects factory.
- `registered_object_t * chFactoryRegisterObject (const char *name, void *objp)`
  Registers a generic object.
- `registered_object_t * chFactoryFindObject (const char *name)`
  Retrieves a registered object.
- `registered_object_t * chFactoryFindObjectByPointer (void *objp)`
  Retrieves a registered object by pointer.
- `void chFactoryReleaseObject (registered_object_t *rop)`
  Releases a registered object.
- `dyn_buffer_t * chFactoryCreateBuffer (const char *name, size_t size)`
  Creates a generic dynamic buffer object.
- `dyn_buffer_t * chFactoryFindBuffer (const char *name)`
  Retrieves a dynamic buffer object.
- `void chFactoryReleaseBuffer (dyn_buffer_t *dbp)`
  Releases a dynamic buffer object.
- `dyn_semaphore_t * chFactoryCreateSemaphore (const char *name, cnt_t n)`
  Creates a dynamic semaphore object.
- `dyn_semaphore_t * chFactoryFindSemaphore (const char *name)`
  Retrieves a dynamic semaphore object.
- `void chFactoryReleaseSemaphore (dyn_semaphore_t *dsp)`
  Releases a dynamic semaphore object.
- `dyn_mailbox_t * chFactoryCreateMailbox (const char *name, size_t n)`
  Creates a dynamic mailbox object.
- `dyn_mailbox_t * chFactoryFindMailbox (const char *name)`
  Retrieves a dynamic mailbox object.
- `void chFactoryReleaseMailbox (dyn_mailbox_t *dmp)`
  Releases a dynamic mailbox object.
- `dyn_objects_fifo_t * chFactoryCreateObjectsFIFO (const char *name, size_t objsize, size_t objn, unsigned objalign)`
  Creates a dynamic "objects FIFO" object.
- `dyn_objects_fifo_t * 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 uint8_t * chFactoryGetBuffer (dyn_buffer_t *dbp)`
  Returns the pointer to the inner buffer.
• static semaphore_t * chFactoryGetSemaphore (dyn_semaphore_t *dsp)
  Returns the pointer to the inner semaphore.

• static mailbox_t * chFactoryGetMailbox (dyn_mailbox_t *dmp)
  Returns the pointer to the inner mailbox.

• static objects_fifo_t * chFactoryGetObjectsFIFO (dyn_objects_fifo_t *dofp)
  Returns the pointer to the inner objects FIFO.

• static pipe_t * chFactoryGetPipe (dyn_pipe_t *dpp)
  Returns the pointer to the inner pipe.

Variables

• objects_factory_t ch_factory
  Factory object static instance.

6.52.2 Macro Definition Documentation

6.52.2.1 CH_CFG_FACTORY_MAX_NAMES_LENGTH

#define CH_CFG_FACTORY_MAX_NAMES_LENGTH 8

Maximum length for object names.

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

6.52.2.2 CH_CFG_FACTORY_OBJECTS_REGISTRY

#define CH_CFG_FACTORY_OBJECTS_REGISTRY TRUE

Enables the registry of generic objects.

6.52.2.3 CH_CFG_FACTORY_GENERIC_BUFFERS

#define CH_CFG_FACTORY_GENERIC_BUFFERS TRUE

Enables factory for generic buffers.

6.52.2.4 CH_CFG_FACTORY_SEMAPHORES [1/2]

#define CH_CFG_FACTORY_SEMAPHORES TRUE

Enables factory for semaphores.
6.52.2.5  CH_CFG_FACTORY_SEMAPHORES [2/2]

#define CH_CFG_FACTORY_SEMAPHORES FALSE

Enables factory for semaphores.

6.52.2.6  CH_CFG_FACTORY_MAILBOXES [1/2]

#define CH_CFG_FACTORY_MAILBOXES TRUE

Enables factory for mailboxes.

6.52.2.7  CH_CFG_FACTORY_MAILBOXES [2/2]

#define CH_CFG_FACTORY_MAILBOXES FALSE

Enables factory for mailboxes.

6.52.2.8  CH_CFG_FACTORY_OBJ_FIFOS [1/3]

#define CH_CFG_FACTORY_OBJ_FIFOS TRUE

Enables factory for objects FIFOs.

6.52.2.9  CH_CFG_FACTORY_OBJ_FIFOS [2/3]

#define CH_CFG_FACTORY_OBJ_FIFOS TRUE

Enables factory for objects FIFOs.

6.52.2.10  CH_CFG_FACTORY_OBJ_FIFOS [3/3]

#define CH_CFG_FACTORY_OBJ_FIFOS FALSE

Enables factory for objects FIFOs.


6.52.2.11  CH_CFG_FACTORY_PIPES [1/2]

#define CH_CFG_FACTORY_PIPES TRUE

Enables factory for Pipes.

6.52.2.12  CH_CFG_FACTORY_PIPES [2/2]

#define CH_CFG_FACTORY_PIPES FALSE

Enables factory for Pipes.

6.52.3  Typedef Documentation

6.52.3.1  dyn_element_t

typedef struct ch_dyn_element dyn_element_t

Type of a dynamic object list element.

6.52.3.2  dyn_list_t

typedef struct ch_dyn_list dyn_list_t

Type of a dynamic object list.

6.52.3.3  registered_object_t

typedef struct ch_registered_static_object registered_object_t

Type of a registered object.

6.52.3.4  dyn_buffer_t

typedef struct ch_dyn_object dyn_buffer_t

Type of a dynamic buffer object.
6.52.3.5 dyn_semaphore_t

typedef struct ch_dyn_semaphore dyn_semaphore_t

Type of a dynamic semaphore.

6.52.3.6 dyn_mailbox_t

typedef struct ch_dyn_mailbox dyn_mailbox_t

Type of a dynamic buffer object.

6.52.3.7 dyn_objects_fifo_t

typedef struct ch_dyn_objects_fifo dyn_objects_fifo_t

Type of a dynamic buffer object.

6.52.3.8 dyn_pipe_t

typedef struct ch_dyn_pipe dyn_pipe_t

Type of a dynamic pipe object.

6.52.3.9 objects_factory_t

typedef struct ch_objects_factory objects_factory_t

Type of the factory main object.

6.52.4 Function Documentation
## 6.52.4.1 __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](image)

## 6.52.4.2 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><code>in</code></td>
<td><code>name</code></td>
<td>name to be assigned to the registered object</td>
</tr>
<tr>
<td><code>in</code></td>
<td><code>objp</code></td>
<td>pointer to the object to be registered</td>
</tr>
</tbody>
</table>

**Returns**

The reference to the registered object.
Return values

| NULL | if the object to be registered cannot be allocated or a registered object with the same name exists. |

Function Class:

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

6.52.4.3 chFactoryFindObject()

```c
registered_object_t * chFactoryFindObject ( 
    const char * name )
```

Retrieves a registered object.

Postcondition

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

Parameters

| in | name | name of the registered object |

Returns

The reference to the found registered object.

Return values

| NULL | if a registered object with the specified name does not exist. |

Function Class:

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

6.52.4.4 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.

6.52.4.5  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.

6.52.4.6  chFactoryCreateBuffer()

```c
dyn_buffer_t * chFactoryCreateBuffer ( 
    const char * name, 
    size_t size )
```

Creates a generic dynamic buffer object.
Postcondition

A reference to the dynamic buffer object is returned and the reference counter is initialized to one.
The dynamic buffer object is filled with zeros.

Parameters

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>name</td>
<td>name to be assigned to the new dynamic buffer object</td>
</tr>
<tr>
<td>in</td>
<td>size</td>
<td>payload size of the dynamic buffer object to be created</td>
</tr>
</tbody>
</table>

Returns

The reference to the created dynamic buffer object.

Return values

**NULL** if the dynamic buffer object cannot be allocated or a dynamic buffer object with the same name exists.

Function Class:

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

6.52.4.7 chFactoryFindBuffer()

```c
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

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>name</td>
<td>name of the dynamic buffer object</td>
</tr>
</tbody>
</table>

Returns

The reference to the found dynamic buffer object.

Return values

**NULL** if a dynamic buffer object with the specified name does not exist.
Function Class:
Normal API, this function can be invoked by regular system threads but not from within a lock zone.

6.52.4.8 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.

6.52.4.9 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

<table>
<thead>
<tr>
<th>in</th>
<th>name</th>
<th>name to be assigned to the new dynamic semaphore object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>n</td>
<td>dynamic semaphore object counter initialization value</td>
</tr>
</tbody>
</table>

Returns

The reference to the created dynamic semaphore object.
Return values

| **NULL** | if the dynamic semaphore object cannot be allocated or a dynamic semaphore with the same name exists. |

Function Class:

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

### 6.52.4.10 chFactoryFindSemaphore()

dyn_semaphore_t * chFactoryFindSemaphore (
  const char * name )

Retrieves a dynamic semaphore object.

**Postcondition**

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

**Parameters**

- **in name**  name of the dynamic semaphore object

**Returns**

The reference to the found dynamic semaphore object.

**Return values**

| **NULL** | if a dynamic semaphore object with the specified name does not exist. |

Function Class:

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

### 6.52.4.11 chFactoryReleaseSemaphore()

void chFactoryReleaseSemaphore ( 
  dyn_semaphore_t * dsp )

Releases a dynamic semaphore object.

The reference counter of the dynamic semaphore object is decreased by one, if reaches zero then the dynamic semaphore object memory is freed.
500 Module Documentation

Parameters

| in | dsp | dynamic semaphore object reference |

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

6.52.4.12 chFactoryCreateMailbox()

dyn_mailbox_t * chFactoryCreateMailbox (  
const char * name,  
size_t n )

Creates a dynamic mailbox object.

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

Parameters

| in | name | name to be assigned to the new dynamic mailbox object |
| in | n | mailbox buffer size as number of messages |

Returns
The reference to the created dynamic mailbox object.

Return values

| NULL | if the dynamic mailbox object cannot be allocated or a dynamic mailbox object with the same name exists. |

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

6.52.4.13 chFactoryFindMailbox()

dyn_mailbox_t * chFactoryFindMailbox (  
const char * name )

Retrieves a dynamic mailbox object.
Postcondition
A reference to the dynamic mailbox object is returned with the reference counter increased by one.

Parameters

| in  | 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.

6.52.4.14 chFactoryReleaseMailbox()

```c
void chFactoryReleaseMailbox (  
    dyn_mailbox_t * dmp )
```

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

Parameters

| in  | dmp | dynamic mailbox object reference |

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

6.52.4.15 chFactoryCreateObjectsFIFO()

```c
dyn_objects_fifo_t * chFactoryCreateObjectsFIFO (  
    const char * name,  
    size_t objsize,  
    size_t objn,  
    unsigned objalign )
```

Creates a dynamic "objects FIFO" object.
Postcondition

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

Parameters

| in | name | name to be assigned to the new dynamic "objects FIFO" object |
| in | objsize | size of objects |
| in | objn | number of objects available |
| in | objalign | required objects alignment |

Returns

The reference to the created dynamic "objects FIFO" object.

Return values

| NULL | if the dynamic "objects FIFO" object cannot be allocated or a dynamic "objects FIFO" object with the same name exists. |

Function Class:

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

6.52.4.16 chFactoryFindObjectsFIFO()

dyn_objects_fifo_t * chFactoryFindObjectsFIFO ( const char * name )

Retrieves a dynamic "objects FIFO" object.

Postcondition

A reference to the dynamic "objects FIFO" object is returned with the reference counter increased by one.

Parameters

| in | name | name of the dynamic "objects FIFO" object |

Returns

The reference to the found dynamic "objects FIFO" object.
Return values

| NULL | if a dynamic "objects FIFO" object with the specified name does not exist. |

Function Class:

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

6.52.4.17 chFactoryReleaseObjectsFIFO()

```c
void chFactoryReleaseObjectsFIFO ( 
    dyn_objects_fifo_t * dofp )
```

Releases a dynamic "objects FIFO" object.

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

Parameters

| in  | dofp  | dynamic "objects FIFO" object reference |

Function Class:

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

6.52.4.18 chFactoryCreatePipe()

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

Creates a dynamic pipe object.

Postcondition

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

The dynamic pipe object is initialized and ready to use.

Parameters

| in  | name  | name to be assigned to the new dynamic pipe object |
| in  | size  | pipe buffer size |
Returns

The reference to the created dynamic pipe object.

Return values

| NULL | if the dynamic pipe object cannot be allocated or a dynamic pipe object with the same name exists.

Function Class:

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

6.52.4.19  chFactoryFindPipe()

dyn_pipe_t * chFactoryFindPipe (  
    const char * name )

Retrieves a dynamic pipe object.

Postcondition

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

Parameters

| in  | name | name of the pipe object |

Returns

The reference to the found dynamic pipe object.

Return values

| NULL | if a dynamic pipe object with the specified name does not exist.

Function Class:

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

6.52.4.20  chFactoryReleasePipe()

void chFactoryReleasePipe (  
    dyn_pipe_t * dpp )
Releases a dynamic pipe object.

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

<table>
<thead>
<tr>
<th></th>
<th>dpp</th>
<th>dynamic pipe object reference</th>
</tr>
</thead>
</table>

Function Class:

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

6.52.4.21 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></th>
<th>dep</th>
<th>pointer to the element field of the object</th>
</tr>
</thead>
</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.

6.52.4.22 chFactoryGetObject()

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

Returns the pointer to the inner registered object.

Parameters

<table>
<thead>
<tr>
<th></th>
<th>rop</th>
<th>registered object reference</th>
</tr>
</thead>
</table>
6.52 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.

6.52.4.23 chFactoryGetBufferSize()

static size_t chFactoryGetBufferSize (  
    dyn_buffer_t * dbp ) [inline], [static]

Returns the size of a generic dynamic buffer object.

Parameters

| in  | 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:

6.52.4.24 chFactoryGetBuffer()

static uint8_t* chFactoryGetBuffer (  
    dyn_buffer_t * dbp ) [inline], [static]

Returns the pointer to the inner buffer.
Parameters

| in | dbp | dynamic buffer object reference |

Returns

The pointer to the dynamic buffer.

Function Class:

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

### 6.52.4.25 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.

### 6.52.4.26 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.

### 6.52.4.27 chFactoryGetObjectsFIFO()

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

Returns the pointer to the inner objects FIFO.

Parameters

| in  | dofp | dynamic "objects FIFO" object reference |

Returns

The pointer to the objects FIFO.

Function Class:

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

### 6.52.4.28 chFactoryGetPipe()

```c
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.
6.52.5 Variable Documentation

6.52.5.1 ch_factory

objects_factory_t ch_factory

Factory object static instance.

Note

It is a global object because it could be accessed through a specific debugger plugin.
Chapter 7

Data Structure Documentation

7.1 ch_binary_semaphore Struct Reference

Binary semaphore type.

#include <chbsem.h>

Inheritance diagram for ch_binary_semaphore:
Collaboration diagram for ch_binary_semaphore:

Additional Inherited Members

7.1.1 Detailed Description

Binary semaphore type.

7.2 ch_delta_list Struct Reference

Delta list element and header structure.
#include <chlists.h>

Collaboration diagram for ch_delta_list:
Data Fields

- **ch_delta_list_t** *next*
  
  Next in the delta list.

- **ch_delta_list_t** *prev*
  
  Previous in the delta list.

- **sysinterval_t** **delta**
  
  Time interval from previous.

### 7.2.1 Detailed Description

Delta list element and header structure.

### 7.2.2 Field Documentation

#### 7.2.2.1 next

```c
ch_delta_list_t* ch_delta_list::next
```

Next in the delta list.

#### 7.2.2.2 prev

```c
ch_delta_list_t* ch_delta_list::prev
```

Previous in the delta list.

#### 7.2.2.3 delta

```c
sysinterval_t ch_delta_list::delta
```

Time interval from previous.
7.3 ch_dyn_element Struct Reference

Type of a dynamic object list element.

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

Collaboration diagram for ch_dyn_element:

![Collaboration Diagram](image)

**Data Fields**

- `struct ch_dyn_element * next

  Next dynamic object in the list.

- `ucnt_t refs`

  Number of references to this object.

### 7.3.1 Detailed Description

Type of a dynamic object list element.

### 7.3.2 Field Documentation

#### 7.3.2.1 next

```c
struct ch_dyn_element* ch_dyn_element::next
```

Next dynamic object in the list.

#### 7.3.2.2 refs

```c
ucnt_t ch_dyn_element::refs
```

Number of references to this object.
7.4 ch_dyn_list Struct Reference

Type of a dynamic object list.

#include <chfactory.h>

Collaboration diagram for ch_dyn_list:

![Collaboration Diagram]

7.4.1 Detailed Description

Type of a dynamic object list.

7.5 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.*

7.5.1 Detailed Description

Type of a dynamic buffer object.

7.5.2 Field Documentation
7.5.2.1 element

dyn_element_t ch_dyn_mailbox::element

List element of the dynamic buffer object.

7.5.2.2 mbx

mailbox_t ch_dyn_mailbox::mbx

The mailbox.

7.6 ch_dyn_object Struct Reference

Type of a dynamic buffer object.

#include <chfactory.h>

Collaboration diagram for ch_dyn_object:

Data Fields

- dyn_element_t element

List element of the dynamic buffer object.
7.6.1 Detailed Description

Type of a dynamic buffer object.

7.6.2 Field Documentation

7.6.2.1 element

dyn_element_t ch_dyn_object::element

List element of the dynamic buffer object.

7.7 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.

7.7.1 Detailed Description

Type of a dynamic buffer object.

7.7.2 Field Documentation

7.7.2.1 element

```c
dyn_element_t ch_dyn_objects_fifo::element
```

List element of the dynamic buffer object.

7.7.2.2 fifo

```c
objects_fifo_t ch_dyn_objects_fifo::fifo
```

The objects FIFO.

7.8 ch_dyn_pipe Struct Reference

Type of a dynamic pipe object.

```c
#include <chfactory.h>
```
Collaboration diagram for ch_dyn_pipe:

Data Fields

- **dyn_element_t element**
  
  *List element of the dynamic pipe object.*

- **pipe_t pipe**
  
  *The pipe.*
7.8.1 Detailed Description
Type of a dynamic pipe object.

7.8.2 Field Documentation

7.8.2.1 element

dyn_element_t ch_dyn_pipe::element
List element of the dynamic pipe object.

7.8.2.2 pipe

pipe_t ch_dyn_pipe::pipe
The pipe.

7.9 ch_dyn_semaphore Struct Reference
Type of a dynamic semaphore.
#include <chfactory.h>
Collaboration diagram for ch_dyn_semaphore:

ChibiOS/RT
Data Fields

- **dyn_element_t element**
  
  List element of the dynamic semaphore.

- **semaphore_t sem**
  
  The semaphore.

### 7.9.1 Detailed Description

Type of a dynamic semaphore.

### 7.9.2 Field Documentation

#### 7.9.2.1 element

```c
dyn_element_t ch_dyn_semaphore::element
```

List element of the dynamic semaphore.

#### 7.9.2.2 sem

```c
semaphore_t ch_dyn_semaphore::sem
```

The semaphore.

### 7.10 ch_job_descriptor Struct Reference

Type of a job descriptor.

```c
#include <chjobs.h>
```

Collaboration diagram for `ch_job_descriptor`:
Data Fields

- `job_function_t jobfunc`
  
  Job function.

- `void * jobarg`
  
  Argument to be passed to the job function.

7.10.1 Detailed Description

Type of a job descriptor.

7.10.2 Field Documentation

7.10.2.1 jobfunc

`job_function_t ch_job_descriptor::jobfunc`

Job function.

7.10.2.2 jobarg

`void* ch_job_descriptor::jobarg`

Argument to be passed to the job function.

7.11 ch_jobs_queue Struct Reference

Type of a jobs queue.

#include <chjobs.h>
Collaboration diagram for ch_jobs_queue:

Data Fields

- **guarded_memory_pool_t free**
  
  Pool of the free jobs.

- **mailbox_t mbx**

  Mailbox of the sent jobs.

### 7.11.1 Detailed Description

Type of a jobs queue.

### 7.11.2 Field Documentation
7.11.2.1 free

guarded_memory_pool_t ch_jobs_queue::free

Pool of the free jobs.

7.11.2.2 mbx

mailbox_t ch_jobs_queue::mbx

Mailbox of the sent jobs.

7.12 ch_list Struct Reference

Structure representing a generic single link list header and element.

#include <chlists.h>

Collaboration diagram for ch_list:

Data Fields

- ch_list_t * next

  Next in the list/queue.

7.12.1 Detailed Description

Structure representing a generic single link list header and element.

7.12.2 Field Documentation
7.12.2.1 next

ch_list_t* ch_list::next

Next in the list/queue.

7.13 ch_mutex Struct Reference

Mutex structure.

#include <chmtx.h>

Collaboration diagram for ch_mutex:

Data Fields

- ch_queue_t queue
  Queue of the threads sleeping on this mutex.

- thread_t * owner
  Owner thread_t pointer or NULL.

- mutex_t * next
  Next mutex_t into an owner-list or NULL.

- cnt_t cnt
  Mutex recursion counter.
7.13.1 Detailed Description

Mutex structure.

7.13.2 Field Documentation

7.13.2.1 queue

ch_queue_t ch_mutex::queue

Queue of the threads sleeping on this mutex.

7.13.2.2 owner

thread_t* ch_mutex::owner

Owner thread_t pointer or NULL.

7.13.2.3 next

mutex_t* ch_mutex::next

Next mutex_t into an owner-list or NULL.

7.13.2.4 cnt

cnt_t ch_mutex::cnt

Mutex recursion counter.
7.14  **ch_objects_cache Struct Reference**

Structure representing a cache object.

```c
#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.

7.14.1 Detailed Description

Structure representing a cache object.

7.14.2 Field Documentation

7.14.2.1 hashn

`ucnt_t ch_objects_cache::hashn`

Number of elements in the hash table.

7.14.2.2 hashp

`oc_hash_header_t * ch_objects_cache::hashp`

Pointer to the hash table.
7.14.2.3  objn

`ucnt_t ch_objects_cache::objn`

Number of elements in the objects table.

7.14.2.4  objsz

`size_t ch_objects_cache::objsz`

Size of elements in the objects table.

7.14.2.5  objvp

`void* ch_objects_cache::objvp`

Pointer to the objects table.

7.14.2.6  lru

`oc_lru_header_t ch_objects_cache::lru`

LRU list header.

7.14.2.7  cache_sem

`semaphore_t ch_objects_cache::cache_sem`

Semaphore for cache access.

7.14.2.8  lru_sem

`semaphore_t ch_objects_cache::lru_sem`

Semaphore for LRU access.
7.15 ch_objects_factory Struct Reference

### 7.14.2.9 readf

**oc_readf_t ch_objects_cache::readf**

Reader functions for cached objects.

### 7.14.2.10 writef

**oc_writef_t ch_objects_cache::writef**

Writer functions for cached objects.

### 7.15 ch_objects_factory Struct Reference

Type of the factory main object.

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

Collaboration diagram for ch_objects_factory:
Data Fields

- **mutex_t mtx**
  
  Factory access mutex or semaphore.

- **dyn_list_t obj_list**
  
  List of the registered objects.

- **memory_pool_t obj_pool**
  
  Pool of the available registered objects.

- **dyn_list_t buf_list**
  
  List of the allocated buffer objects.

- **dyn_list_t sem_list**
  
  List of the allocated semaphores.

- **memory_pool_t sem_pool**
  
  Pool of the available semaphores.

- **dyn_list_t mbx_list**
  
  List of the allocated buffer objects.

- **dyn_list_t fifo_list**
  
  List of the allocated "objects FIFO" objects.

- **dyn_list_t pipe_list**
  
  List of the allocated pipe objects.

### 7.15.1 Detailed Description

Type of the factory main object.

### 7.15.2 Field Documentation

#### 7.15.2.1 mtx

mutex_t ch_objects_factory::mtx

Factory access mutex or semaphore.

#### 7.15.2.2 obj_list

dyn_list_t ch_objects_factory::obj_list

List of the registered objects.
7.15.2.3  **obj_pool**

```cpp
memory_pool_t ch_objects_factory::obj_pool
```

Pool of the available registered objects.

7.15.2.4  **buf_list**

```cpp
dyn_list_t ch_objects_factory::buf_list
```

List of the allocated buffer objects.

7.15.2.5  **sem_list**

```cpp
dyn_list_t ch_objects_factory::sem_list
```

List of the allocated semaphores.

7.15.2.6  **sem_pool**

```cpp
memory_pool_t ch_objects_factory::sem_pool
```

Pool of the available semaphores.

7.15.2.7  **mbx_list**

```cpp
dyn_list_t ch_objects_factory::mbx_list
```

List of the allocated buffer objects.

7.15.2.8  **fifo_list**

```cpp
dyn_list_t ch_objects_factory::fifo_list
```

List of the allocated "objects FIFO" objects.
7.15.2.9 pipe_list

dyn_list_t ch_objects_factory::pipe_list

List of the allocated pipe objects.

7.16 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.
7.16.1 Detailed Description

Type of an objects FIFO.

7.16.2 Field Documentation

7.16.2.1 free

 guarded_memory_pool_t ch_objects_fifo::free

Pool of the free objects.

7.16.2.2 mbx

 mailbox_t ch_objects_fifo::mbx

Mailbox of the sent objects.

7.17 ch_oc_hash_header Struct Reference

Structure representing an hash table element.

#include <chobjcaches.h>
Collaboration diagram for ch_oc_hash_header:

Data Fields

- oc_object_t * hash_next
  
  Next in the collisions list.

- oc_object_t * hash_prev
  
  Previous in the collisions list.

7.17.1 Detailed Description

Structure representing an hash table element.

7.17.2 Field Documentation
7.17.2.1 hash_next

\texttt{oc_object_t* \ ch\_oc\_hash\_header::hash\_next}

Next in the collisions list.

7.17.2.2 hash_prev

\texttt{oc_object_t* \ ch\_oc\_hash\_header::hash\_prev}

Previous in the collisions list.

7.18 \texttt{ch\_oc\_lru\_header} Struct Reference

Structure representing an hash table element.

\texttt{#include <chobjcaches.h>}

Collaboration diagram for \texttt{ch\_oc\_lru\_header}:
Data Fields

- `oc_object_t * hash_next
  Next in the collisions list.
- `oc_object_t * hash_prev
  Previous in the collisions list.
- `oc_object_t * lru_next
  Next in the LRU list.
- `oc_object_t * lru_prev
  Previous in the LRU list.

7.18.1 Detailed Description

Structure representing an hash table element.

7.18.2 Field Documentation

7.18.2.1 hash_next

`oc_object_t* ch_oc_lru_header::hash_next
Next in the collisions list.

7.18.2.2 hash_prev

`oc_object_t* ch_oc_lru_header::hash_prev
Previous in the collisions list.

7.18.2.3 lru_next

`oc_object_t* ch_oc_lru_header::lru_next
Next in the LRU list.

7.18.2.4 lru_prev

`oc_object_t* ch_oc_lru_header::lru_prev
Previous in the LRU list.
7.19 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.
Semaphore for object access.

- **oc_flags_t obj_flags**
  Object flags.
- **void * dptr**
  User pointer.

### 7.19.1 Detailed Description

Structure representing a cached object.

### 7.19.2 Field Documentation

#### 7.19.2.1 hash_next

\[
\text{oc_object_t* ch_oc_object::hash_next}
\]

Next in the collisions list.

#### 7.19.2.2 hash_prev

\[
\text{oc_object_t* ch_oc_object::hash_prev}
\]

Previous in the collisions list.

#### 7.19.2.3 lru_next

\[
\text{oc_object_t* ch_oc_object::lru_next}
\]

Next in the LRU list.

#### 7.19.2.4 lru_prev

\[
\text{oc_object_t* ch_oc_object::lru_prev}
\]

Previous in the LRU list.
7.19.2.5 obj_group

```
uint32_t ch_oc_object::obj_group
```

Object group.

7.19.2.6 obj_key

```
uint32_t ch_oc_object::obj_key
```

Object key.

7.19.2.7 obj_sem

```
semaphore_t ch_oc_object::obj_sem
```

Semaphore for object access.

7.19.2.8 obj_flags

```
oc_flags_t ch_oc_object::obj_flags
```

Object flags.

7.19.2.9 dptr

```
void* ch_oc_object::dptr
```

User pointer.

**Note**

This pointer can be used to refer to external buffers, `chCacheObjectInit()` initializes it to `NULL`.
# 7.20 ch_os_instance Struct Reference

System instance data structure.

```
#include <chobjects.h>
```

Collaboration diagram for ch_os_instance:

![Collaboration Diagram](image)

## Data Fields

- **ready_list_t rlist**  
  Ready list header.
- **virtual_timers_list_t vtlist**  
  Virtual timers delta list header.
- **registry_t reglist**  
  Registry header.
- **core_id_t core_id**  
  Core associated to this instance.
- **rfcu_t rfcu**  
  Runtime Faults Collection Unit for this instance.
- **const os_instance_config_t * config**  
  Pointer to the instance configuration data.
- **thread_t mainthread**  
  Main thread descriptor.
- **system_debug_t dbg**
7.20 ch_os_instance Struct Reference

System debug.

- trace_buffer_t trace_buffer
  Trace buffer.
- kernel_stats_t kernel_stats
  Global kernel statistics.

7.20.1 Detailed Description

System instance data structure.

7.20.2 Field Documentation

7.20.2.1 rlist

ready_list_t ch_os_instance::rlist

Ready list header.

7.20.2.2 vtlist

virtual_timers_list_t ch_os_instance::vtlist

Virtual timers delta list header.

7.20.2.3 reglist

registry_t ch_os_instance::reglist

Registry header.

Note

This field is present only if the SMP mode is disabled.

7.20.2.4 core_id

core_id_t ch_os_instance::core_id

Core associated to this instance.
7.20.2.5 rfcu

rfcu_t ch_os_instance::rfcu

Runtime Faults Collection Unit for this instance.

Note

This field is present only if the SMP mode is disabled.

7.20.2.6 config

const os_instance_config_t* ch_os_instance::config

Pointer to the instance configuration data.

7.20.2.7 mainthread

thread_t ch_os_instance::mainthread

Main thread descriptor.

7.20.2.8 dbg

system_debug_t ch_os_instance::dbg

System debug.

7.20.2.9 trace_buffer

trace_buffer_t ch_os_instance::trace_buffer

Trace buffer.

7.20.2.10 kernel_stats

kernel_stats_t ch_os_instance::kernel_stats

Global kernel statistics.
7.21 ch_os_instance_config Struct Reference

Type of an system instance configuration.

#include <chobjects.h>

Collaboration diagram for ch_os_instance_config:

---

Data Fields

- **const char ∗ name**
  
  *Instance name.*

- **stkalign_t ∗ mainthread_base**
  
  *Lower limit of the main function thread stack.*

- **stkalign_t ∗ mainthread_end**
  
  *Upper limit of the main function thread stack.*

- **stkalign_t ∗ idlethread_base**
  
  *Lower limit of the dedicated idle thread stack.*

- **stkalign_t ∗ idlethread_end**
  
  *Upper limit of the dedicated idle thread stack.*

---

7.21.1 Detailed Description

Type of an system instance configuration.

7.21.2 Field Documentation

7.21.2.1 name

**const char ∗ ch_os_instance_config::name**

*Instance name.*
7.21.2.2 mainthread_base

stkalign_t* ch_os_instance_config::mainthread_base

Lower limit of the main function thread stack.

7.21.2.3 mainthread_end

stkalign_t* ch_os_instance_config::mainthread_end

Upper limit of the main function thread stack.

7.21.2.4 idlethread_base

stkalign_t* ch_os_instance_config::idlethread_base

Lower limit of the dedicated idle thread stack.

7.21.2.5 idlethread_end

stkalign_t* ch_os_instance_config::idlethread_end

Upper limit of the dedicated idle thread stack.

7.22 ch_priority_queue Struct Reference

Structure representing a generic priority-ordered bidirectional linked list header and element.

#include <chlists.h>

Collaboration diagram for ch_priority_queue:
7.22 ch_priority_queue Struct Reference

Data Fields

- ch_priority_queue_t * next
  
  Next in the queue.

- ch_priority_queue_t * prev
  
  Previous in the queue.

- tprio_t prio
  
  Priority of this element.

7.22.1 Detailed Description

Structure representing a generic priority-ordered bidirectional linked list header and element.

Note

Link fields are void pointers in order to avoid aliasing issues.

7.22.2 Field Documentation

7.22.2.1 next

ch_priority_queue_t* ch_priority_queue::next

Next in the queue.

7.22.2.2 prev

ch_priority_queue_t* ch_priority_queue::prev

Previous in the queue.

7.22.2.3 prio

tprio_t ch_priority_queue::prio

Priority of this element.
7.23  ch_queue Struct Reference

Structure representing a generic bidirectional linked list header and element.

#include <chlists.h>

Collaboration diagram for ch_queue:

```
ch_queue
 +prev
+next
```

Data Fields

- `ch_queue_t * next`
  Next in the list/queue.

- `ch_queue_t * prev`
  Previous in the queue.

7.23.1 Detailed Description

Structure representing a generic bidirectional linked list header and element.

7.23.2 Field Documentation

7.23.2.1 next

`ch_queue_t* ch_queue::next`

Next in the list/queue.
7.23.2.2 prev

ch_queue_t* ch_queue::prev

Previous in the queue.

7.24 ch_ready_list Struct Reference

Type of a ready list header.

#include <chobjects.h>

Collaboration diagram for ch_ready_list:

Data Fields

- ch_priority_queue_t pqueue
  Threads ordered queues header.
- thread_t * current
  The currently running thread.
7.24.1 Detailed Description

Type of a ready list header.

7.24.2 Field Documentation

7.24.2.1 pqueue

\texttt{ch\_priority\_queue\_t\ ch\_ready\_list::pqueue}

Threads ordered queues header.

Note

The priority field must be initialized to zero.

7.24.2.2 current

\texttt{thread\_t\*\ ch\_ready\_list::current}

The currently running thread.

7.25 \texttt{ch\_registered\_static\_object} Struct Reference

Type of a registered object.

\#include \texttt{<chfactory.h>}

Collaboration diagram for \texttt{ch\_registered\_static\_object}:

\begin{center}
\begin{tikzpicture}
  \node (ch_reg) at (0,0) {
    \texttt{ch\_registered\_static\_object}
  };
  \node (ch_dyn) at (0,-2) {
    \texttt{ch\_dyn\_element}
  };
  \node (obj) at (0,-4) {
    \texttt{objp}
  };
  \node (ref) at (0,-5) {
    \texttt{refs}
  };
  \node (name) at (0,-6) {
    \texttt{name}
  };
  \node (element) at (0,-7) {
    \texttt{element}
  };
  \node (next) at (0,-8) {
    \texttt{next}
  };
  \draw (ch_reg) -- (ch_dyn);
  \draw (ch_reg) -- (obj);
  \draw (ch_reg) -- (ref);
  \draw (ch_reg) -- (name);
  \draw (ch_dyn) -- (element);
  \draw (element) -- (next);
\end{tikzpicture}
\end{center}
Data Fields

- **dyn_element_t element**
  
  List element of the registered object.

- **void * objp**
  
  Pointer to the object.

### 7.25.1 Detailed Description

Type of a registered object.

### 7.25.2 Field Documentation

#### 7.25.2.1 element

*`dyn_element_t ch_registered_static_object::element`*

List element of the registered object.

#### 7.25.2.2 objp

*`void* ch_registered_static_object::objp`*

Pointer to the object.

**Note**

The type of the object is not stored in anyway.
7.26  ch_registry Struct Reference

Type of a registry structure.

#include <chobjects.h>

Collaboration diagram for ch_registry:

Data Fields

- ch_queue_t queue
  
  Registry queue header.

7.26.1  Detailed Description

Type of a registry structure.

7.26.2  Field Documentation

7.26.2.1  queue

ch_queue_t ch_registry::queue

Registry queue header.
Type of an RFCU structure.

#include <chrfcu.h>

Collaboration diagram for ch_rfcu:

```
ch_rfcu
+ mask
```

Data Fields

- `rfcu_mask_t mask`
  
  Mask of the pending runtime faults.

7.27.1 Detailed Description

Type of an RFCU structure.

7.27.2 Field Documentation

7.27.2.1 mask

```
rfcu_mask_t ch_rfcu::mask
```

Mask of the pending runtime faults.
7.28  ch_semaphore Struct Reference

Semaphore structure.

#include <chsem.h>

Inheritance diagram for ch_semaphore:

Collaboration diagram for ch_semaphore:

Data Fields

- ch_queue_t queue
Queue of the threads sleeping on this semaphore.

- cnt_t cnt
  The semaphore counter.

### 7.28.1 Detailed Description

Semaphore structure.

### 7.28.2 Field Documentation

#### 7.28.2.1 queue

ch_queue_t ch_semaphore::queue

Queue of the threads sleeping on this semaphore.

#### 7.28.2.2 cnt

cnt_t ch_semaphore::cnt

The semaphore counter.

### 7.29 ch_system Struct Reference

Type of system data structure.

#include <chobjects.h>
Data Structure Documentation

Collaboration diagram for ch_system:

---

Data Fields

- **system_state_t state**
  
  Operating system state.

- **os_instance_t * instances [PORT_CORES_NUMBER]**
  
  Initialized OS instances or **NULL**.

- **tm_calibration_t tmc**
  
  Time measurement calibration data.

- **registry_t reglist**
  
  Registry header.

- **rfcu_t rfcu**
  
  Runtime Faults Collection Unit.

---

### 7.29.1 Detailed Description

Type of system data structure.

---

### 7.29.2 Field Documentation
7.29.2.1  state

```c
system_state_t ch_system::state
```

Operating system state.

7.29.2.2  instances

```c
os_instance_t* ch_system::instances[PORT_CORES_NUMBER]
```

Initialized OS instances or NULL.

7.29.2.3  tmc

```c
tm_calibration_t ch_system::tmc
```

Time measurement calibration data.

7.29.2.4  reglist

```c
registry_t ch_system::reglist
```

Registry header.

Note

This field is present only if the SMP mode is enabled.

7.29.2.5  rfcu

```c
rfcu_t ch_system::rfcu
```

Runtime Faults Collection Unit.

Note

This field is present only if the SMP mode is enabled.
7.30  ch_system_debug Struct Reference

System debug data structure.

#include <chdebug.h>

Collaboration diagram for ch_system_debug:

![Collaboration Diagram]

**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.

7.30.1  Detailed Description

System debug data structure.

7.30.2  Field Documentation

7.30.2.1  panic_msg

    const char ∗volatile ch_system_debug::panic_msg

    Pointer to the panic message.

This pointer is meant to be accessed through the debugger, it is written once and then the system is halted.

**Note**

Accesses to this pointer must never be optimized out so the field itself is declared volatile.
7.30.2.2 isr_cnt

cnt_t ch_system_debug::isr_cnt

ISR nesting level.

7.30.2.3 lock_cnt

cnt_t ch_system_debug::lock_cnt

Lock nesting level.

7.31 ch_thread Struct Reference

Structure representing a thread.

#include <chobjects.h>

Collaboration diagram for ch_thread:
Data Fields

• union {
    ch_list_t list
    Threads lists element.
    ch_queue_t queue
    Threads queues element.
    ch_priority_queue_t pqueue
    Threads ordered queues element.
} hdr

    Shared list headers.
• struct port_context ctx
    Processor context.
• ch_queue_t rqueue
    Registry queue element.
• os_instance_t * owner
    OS instance owner of this thread.
• const char * name
    Thread name or NULL.
• stkalign_t * wabase
    Working area base address.
• tstate_t state
    Current thread state.
• tmode_t flags
    Various thread flags.
• trefs_t refs
    References to this thread.
• tslices_t ticks
    Number of ticks remaining to this thread.
• volatile systime_t time
    Thread consumed time in ticks.
• union {
    msg_t rdymsg
    Thread wakeup code.
    msg_t exitcode
    Thread exit code.
    void * wtobjp
    Pointer to a generic "wait" object.
    thread_reference_t * wttrp
    Pointer to a generic thread reference object.
    msg_t sentmsg
    Thread sent message.
    struct ch_semaphore * wtsemp
    Pointer to a generic semaphore object.
    struct ch_mutex * wtmtxp
    Pointer to a generic mutex object.
    eventmask_t ewmask
    Enabled events mask.
} u

    State-specific fields.
• ch_list_t waiting
    Termination waiting list.
• ch_queue_t msgqueue
  Messages queue.
• eventmask_t 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.

7.31.1 Detailed Description

Structure representing a thread.

Note
Not all the listed fields are always needed, by switching off some not needed ChibiOS/RT subsystems it is possible to save RAM space by shrinking this structure.

7.31.2 Field Documentation

7.31.2.1 list

ch_list_t ch_thread::list

Threads lists element.

7.31.2.2 queue

ch_queue_t ch_thread::queue

Threads queues element.

7.31.2.3 pqueue

ch_priority_queue_t ch_thread::pqueue

Threads ordered queues element.
7.31.2.4  hdr

union { ... } ch_thread::hdr

Shared list headers.

7.31.2.5  ctx

struct port_context ch_thread::ctx

Processor context.

7.31.2.6  rqueue

ch_queue_t ch_thread::rqueue

Registry queue element.

7.31.2.7  owner

os_instance_t* ch_thread::owner

OS instance owner of this thread.

7.31.2.8  name

const char* ch_thread::name

Thread name or NULL.

7.31.2.9  wabase

stkalign_t* ch_thread::wabase

Working area base address.

Note

This pointer is used for stack overflow checks and for dynamic threading.
7.31.2.10  state

tstate_t ch_thread::state

Current thread state.

7.31.2.11  flags

tmode_t ch_thread::flags

Various thread flags.

7.31.2.12  refs

trefs_t ch_thread::refs

References to this thread.

7.31.2.13  ticks

tslices_t ch_thread::ticks

Number of ticks remaining to this thread.

7.31.2.14  time

volatile systime_t ch_thread::time

Thread consumed time in ticks.

Note
This field can overflow.
7.31.2.15  rdymsg

msg_t ch_thread::rdymsg

Thread wakeup code.

Note
This field contains the low level message sent to the thread by the waking thread or interrupt handler. The value is valid after exiting the chSchWakeupS() function.

7.31.2.16  exitcode

msg_t ch_thread::exitcode

Thread exit code.

Note
The thread termination code is stored in this field in order to be retrieved by the thread performing a chThdWait() on this thread.

7.31.2.17  wtobjp

void* ch_thread::wtobjp

Pointer to a generic "wait" object.

Note
This field is used to get a generic pointer to a synchronization object and is valid when the thread is in one of the wait states.

7.31.2.18  wttrp

thread_reference_t* ch_thread::wttrp

Pointer to a generic thread reference object.

Note
This field is used to get a pointer to a synchronization object and is valid when the thread is in CH_STATE←_SUSPENDED state.
7.31.2.19 sentmsg

```c
msg_t ch_thread::sentmsg
```

Thread sent message.

7.31.2.20 wtsem

```c
struct ch_semaphore* ch_thread::wtsem
```

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.

7.31.2.21 wtmtx

```c
struct ch_mutex* ch_thread::wtmtx
```

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.

7.31.2.22 ewmask

```c
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.
7.31.2.23 u

union { ... } ch_thread::u

State-specific fields.

Note

All the fields declared in this union are only valid in the specified state or condition and are thus volatile.

7.31.2.24 waiting

ch_list_t ch_thread::waiting

Termination waiting list.

7.31.2.25 msgqueue

ch_queue_t ch_thread::msgqueue

Messages queue.

7.31.2.26 epending

eventmask_t ch_thread::epending

Pending events mask.

7.31.2.27 mtxlist

struct ch_mutex* ch_thread::mtxlist

List of the mutexes owned by this thread.

Note

The list is terminated by a NULL in this field.
### 7.32 ch_threads_queue Struct Reference

Type of a threads queue.

```c
#include <chobjects.h>
```

**Collaboration diagram for ch_threads_queue:**

- **ch_queue**
  - `next` + `prev`
  - `+queue`
- **ch_threads_queue**
Data Fields

- `ch_queue_t queue`
  
  Threads queue header.

7.32.1 Detailed Description

Type of a threads queue.

7.32.2 Field Documentation

7.32.2.1 queue

`ch_queue_t ch_threads_queue::queue`

Threads queue header.

7.33 ch_virtual_timer Struct Reference

Structure representing a Virtual Timer.

#include <chobjects.h>

Collaboration diagram for ch_virtual_timer:
7.33 ch_virtual_timer Struct Reference

Data Fields

- `ch_delta_list_t dlist`
  Delta list element.
- `vtfunc_t func`
  Timer callback function pointer.
- `void * par`
  Timer callback function parameter.
- `sysinterval_t reload`
  Current reload interval.

7.33.1 Detailed Description

Structure representing a Virtual Timer.

7.33.2 Field Documentation

7.33.2.1 dlist

`ch_delta_list_t ch_virtual_timer::dlist`

Delta list element.

7.33.2.2 func

`vtfunc_t ch_virtual_timer::func`

Timer callback function pointer.

7.33.2.3 par

`void * ch_virtual_timer::par`

Timer callback function parameter.

7.33.2.4 reload

`sysinterval_t ch_virtual_timer::reload`

Current reload interval.
7.34 ch_virtual_timers_list Struct Reference

Type of virtual timers list header.

#include <chobjects.h>

Collaboration diagram for ch_virtual_timers_list:

Data Fields

- ch_delta_list_t dlist
  
  Delta list header.

- volatile systime_t systime
  
  System Time counter.

- systime_t lasttime
  
  System time of the last tick event.

- volatile uint64_t laststamp
  
  Last generated time stamp.

7.34.1 Detailed Description

Type of virtual timers list header.

Note

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

7.34.2.1 dlist

\texttt{ch_delta_list_t ch\_virtual\_timers\_list::dlist}

Delta list header.

7.34.2.2 systime

\texttt{volatile systime\_t ch\_virtual\_timers\_list::systime}

System Time counter.

7.34.2.3 lasttime

\texttt{systime\_t ch\_virtual\_timers\_list::lasttime}

System time of the last tick event.

7.34.2.4 laststamp

\texttt{volatile uint64\_t ch\_virtual\_timers\_list::laststamp}

Last generated time stamp.
7.35  chdebug_t Struct Reference

ChibiOS/RT memory signature record.

#include <chregistry.h>

Collaboration diagram for chdebug_t:

Data Fields

- char identifier [4]
  Always set to "main".

- uint8_t zero
  Must be zero.

- uint8_t size
  Size of this structure.

- uint16_t version
  Encoded ChibiOS/RT version.

- uint8_t ptrsize
  Size of a pointer.

- uint8_t timesize
  Size of a systime_t.

- uint8_t threadsize
  Size of a thread_t.

- uint8_t off_prio
  Offset of prio field.

- uint8_t off_ctx
  and 23 more...
Offset of `ctx` field.

- `uint8_t off_newer`
  Offset of `newer` field.

- `uint8_t off_older`
  Offset of `older` field.

- `uint8_t off_name`
  Offset of `name` field.

- `uint8_t off_stklimit`
  Offset of `stklimit` field.

- `uint8_t off_state`
  Offset of `state` field.

- `uint8_t off_flags`
  Offset of `flags` field.

- `uint8_t off_refs`
  Offset of `refs` field.

- `uint8_t off_preempt`
  Offset of `ticks` field.

- `uint8_t off_time`
  Offset of `time` field.

- `uint8_t intctxsize`
  Size of a `port_intctx`.

- `uint8_t intervalsize`
  Size of a `sysinterval_t`.

- `uint8_t instancesnum`
  Number of instances.

- `uint8_t off_sys_state`
  Offset of `state` field.

- `uint8_t off_sys_instances`
  Offset of `instances` array field.

- `uint8_t off_sys_reglist`
  Offset of `reglist` field.

- `uint8_t off_sys_rfcu`
  Offset of `rfcu` field.

- `uint8_t off_inst_rlist_current`
  Offset of `rlist.current` field.

- `uint8_t off_inst_rlist`
  Offset of `rlist` field.

- `uint8_t off_inst_vtlist`
Offset of `vtlist` field.

- `uint8_t off_inst_reglist`
  Offset of `reglist` field.
- `uint8_t off_inst_core_id`
  Offset of `core_id` field.
- `uint8_t off_inst_rfcu`
  Offset of `rfcu` field.

### 7.35.1 Detailed Description

ChibiOS/RT memory signature record.

### 7.35.2 Field Documentation

#### 7.35.2.1 `identifier`

```c
char chdebug_t::identifier[4]
```

Always set to "main".

#### 7.35.2.2 `zero`

```c
uint8_t chdebug_t::zero
```

Must be zero.

#### 7.35.2.3 `size`

```c
uint8_t chdebug_t::size
```

Size of this structure.
7.35.2.4 version

uint16_t chdebug_t::version

Encoded ChibiOS/RT version.

7.35.2.5 ptrsize

uint8_t chdebug_t::ptrsize

Size of a pointer.

7.35.2.6 timesize

uint8_t chdebug_t::timesize

Size of a systime_t.

7.35.2.7 threadsize

uint8_t chdebug_t::threadsize

Size of a thread_t.

7.35.2.8 off_prio

uint8_t chdebug_t::off_prio

Offset of prio field.

7.35.2.9 off_ctx

uint8_t chdebug_t::off_ctx

Offset of ctx field.
7.35.2.10  off_newer

uint8_t chdebug_t::off_newer
Offset of newer field.

7.35.2.11  off_older

uint8_t chdebug_t::off_older
Offset of older field.

7.35.2.12  off_name

uint8_t chdebug_t::off_name
Offset of name field.

7.35.2.13  off_stklimit

uint8_t chdebug_t::off_stklimit
Offset of stklimit field.

7.35.2.14  off_state

uint8_t chdebug_t::off_state
Offset of state field.

7.35.2.15  off_flags

uint8_t chdebug_t::off_flags
Offset of flags field.
7.35.2.16  off_refs

uint8_t chdebug_t::off_refs
Offset of refs field.

7.35.2.17  off_preempt

uint8_t chdebug_t::off_preempt
Offset of ticks field.

7.35.2.18  off_time

uint8_t chdebug_t::off_time
Offset of time field.

7.35.2.19  intctxsize

uint8_t chdebug_t::intctxsize
Size of a port_intctx.

7.35.2.20  intervalsize

uint8_t chdebug_t::intervalsize
Size of a sysinterval_t.

7.35.2.21  instancesnum

uint8_t chdebug_t::instancesnum
Number of instances.
7.35.2.22  off_sys_state

uint8_t chdebug_t::off_sys_state
Offset of state field.

7.35.2.23  off_sys_instances

uint8_t chdebug_t::off_sys_instances
Offset of instances array field.

7.35.2.24  off_sys_reglist

uint8_t chdebug_t::off_sys_reglist
Offset of reglist field.

7.35.2.25  off_sys_rfcu

uint8_t chdebug_t::off_sys_rfcu
Offset of rfcu field.

7.35.2.26  off_inst_rlist_current

uint8_t chdebug_t::off_inst_rlist_current
Offset of rlist.current field.

7.35.2.27  off_inst_rlist

uint8_t chdebug_t::off_inst_rlist
Offset of rlist field.
7.35.2.28 off_inst_vtlist

uint8_t chdebug_t::off_inst_vtlist
Offset of vtlist field.

7.35.2.29 off_inst_reglist

uint8_t chdebug_t::off_inst_reglist
Offset of reglist field.

7.35.2.30 off_inst_core_id

uint8_t chdebug_t::off_inst_core_id
Offset of core_id field.

7.35.2.31 off_inst_rfcu

uint8_t chdebug_t::off_inst_rfcu
Offset of rfcu field.

7.36 condition_variable Struct Reference

condition_variable_t structure.
#include <chcond.h>
Collaboration diagram for condition_variable:
Data Fields

- ch_queue_t queue

  Condition variable threads queue.

7.36.1 Detailed Description

condition_variable_t structure.

7.36.2 Field Documentation

7.36.2.1 queue

ch_queue_t condition_variable::queue

Condition variable threads queue.

7.37 event_listener Struct Reference

Event Listener structure.

#include <chevents.h>

Collaboration diagram for event_listener:
Data Fields

- \texttt{event_listener\_t * next}
  \textit{Next Event Listener registered on the event source.}

- \texttt{thread\_t * listener}
  \textit{Thread interested in the event source.}

- \texttt{eventmask\_t events}
  \textit{Events to be set in the listening thread.}

- \texttt{eventflags\_t flags}
  \textit{Flags added to the listener by the event source.}

- \texttt{eventflags\_t wflags}
  \textit{Flags that this listener interested in.}

7.37.1 Detailed Description

Event Listener structure.

7.37.2 Field Documentation

7.37.2.1 next

\texttt{event\_listener\_t* event\_listener::next}

Next Event Listener registered on the event source.

7.37.2.2 listener

\texttt{thread\_t* event\_listener::listener}

Thread interested in the event source.

7.37.2.3 events

\texttt{eventmask\_t event\_listener::events}

Events to be set in the listening thread.
7.37.2.4 **flags**

```
eventflags_t event_listener::flags
```

Flags added to the listener by the event source.

7.37.2.5 **wflags**

```
eventflags_t event_listener::wflags
```

Flags that this listener interested in.

7.38 **event_source Struct Reference**

Event Source structure.

```c
#include <chevents.h>
```
7.38 event_source Struct Reference

Collaboration diagram for event_source:

Data Fields

- `event_listener_t * next`

  First Event Listener registered on the Event Source.

7.38.1 Detailed Description

Event Source structure.

7.38.2 Field Documentation
7.38.2.1 next

`event_listener_t* event_source::next`

First Event Listener registered on the Event Source.

7.39 guarded_memory_pool_t Struct Reference

Guarded memory pool descriptor.

#include `<chmempools.h>`

Collaboration diagram for guarded_memory_pool_t:
7.39.1 Detailed Description

Guarded memory pool descriptor.

7.39.2 Field Documentation

7.39.2.1 sem

semaphore_t guarded_memory_pool_t::sem

Counter semaphore guarding the memory pool.

7.39.2.2 pool

memory_pool_t guarded_memory_pool_t::pool

The memory pool itself.

7.40 heap_header Union Reference

Memory heap block header.

#include <chmemheaps.h>

Collaboration diagram for heap_header:
7.40.1 Detailed Description

Memory heap block header.

7.40.2 Field Documentation

7.40.2.1 next

`heap_header_t* heap_header::next`

Next block in free list.

7.40.2.2 pages

`size_t heap_header::pages`

Size of the area in pages.

7.40.2.3 heap

`memory_heap_t* heap_header::heap`

Block owner heap.

7.40.2.4 size

`size_t heap_header::size`

Size of the area in bytes.
7.41 kernel_stats_t Struct Reference

Type of a kernel statistics structure.

#include <chstats.h>

Collaboration diagram for kernel_stats_t:

```
Data Fields

- ucnt_t n_irq
  Number of IRQs.

- ucnt_t n_ctxswc
  Number of context switches.

- time_measurement_t m_crit_thd
  Measurement of threads critical zones duration.

- time_measurement_t m_crit_isr
  Measurement of ISRs critical zones duration.

7.41.1 Detailed Description

Type of a kernel statistics structure.
### 7.41.2 Field Documentation

#### 7.41.2.1 n_irq

`ucnt_t kernel_stats_t::n_irq`

Number of IRQs.

#### 7.41.2.2 n_ctxswc

`ucnt_t kernel_stats_t::n_ctxswc`

Number of context switches.

#### 7.41.2.3 m_crit_thd

`time_measurement_t kernel_stats_t::m_crit_thd`

Measurement of threads critical zones duration.

#### 7.41.2.4 m_crit_isr

`time_measurement_t kernel_stats_t::m_crit_isr`

Measurement of ISRs critical zones duration.
7.42 mailbox_t Struct Reference

Structure representing a mailbox object.

#include <chmboxes.h>

Collaboration diagram for mailbox_t:

Data Fields

- **msg_t** = buffer
  
  Pointer to the mailbox buffer.

- **msg_t** = top
  
  Pointer to the location after the buffer.

- **msg_t** = wrptr
  
  Write pointer.

- **msg_t** = rdptr
  
  Read pointer.
7.42.1 Detailed Description

Structure representing a mailbox object.

7.42.2 Field Documentation

7.42.2.1 buffer

`msg_t* mailbox_t::buffer`

Pointer to the mailbox buffer.

7.42.2.2 top

`msg_t* mailbox_t::top`

Pointer to the location after the buffer.

7.42.2.3 wrptr

`msg_t* mailbox_t::wrptr`

Write pointer.
7.42.2.4 rdptr

`msg_t* mailbox_t::rdptr`

Read pointer.

7.42.2.5 cnt

`size_t mailbox_t::cnt`

Messages in queue.

7.42.2.6 reset

`bool mailbox_t::reset`

True in reset state.

7.42.2.7 qw

`threads_queue_t mailbox_t::qw`

Queued writers.

7.42.2.8 qr

`threads_queue_t mailbox_t::qr`

Queued readers.
7.43 memcore_t Struct Reference

Type of memory core object.

#include <chmemcore.h>

Collaboration diagram for memcore_t:

![Collaboration diagram for memcore_t](image)

**Data Fields**

- `uint8_t* basemem`
  
  *Next free address.*

- `uint8_t* topmem`
  
  *Final address.*

7.43.1 Detailed Description

Type of memory core object.

7.43.2 Field Documentation

7.43.2.1 basemem

`uint8_t* memcore_t::basemem`

Next free address.

7.43.2.2 topmem

`uint8_t* memcore_t::topmem`

Final address.
7.44 memory_heap Struct Reference

Structure describing a memory heap.

#include <chmemheaps.h>

Collaboration diagram for memory_heap:
Data Fields

- `memgetfunc2_t provider`
  Memory blocks provider for this heap.

- `heap_header_t header`
  Free blocks list header.

- `mutex_t mtx`
  Heap access mutex.

7.44.1 Detailed Description

Structure describing a memory heap.

7.44.2 Field Documentation

7.44.2.1 provider

`memgetfunc2_t memory_heap::provider`

Memory blocks provider for this heap.

7.44.2.2 header

`heap_header_t memory_heap::header`

Free blocks list header.

7.44.2.3 mtx

`mutex_t memory_heap::mtx`

Heap access mutex.
7.45 memory_pool_t Struct Reference

Memory pool descriptor.

#include <chmempools.h>

Collaboration diagram for memory_pool_t:

Data Fields

• struct pool_header * next
  Pointer to the header.

• size_t object_size
  Memory pool objects size.

• unsigned align
  Required alignment.

• memgetfunc_t provider
  Memory blocks provider for this pool.

7.45.1 Detailed Description

Memory pool descriptor.

7.45.2 Field Documentation
7.45.2.1 next

struct pool_header* memory_pool_t::next

Pointer to the header.

7.45.2.2 object_size

size_t memory_pool_t::object_size

Memory pool objects size.

7.45.2.3 align

unsigned memory_pool_t::align

Required alignment.

7.45.2.4 provider

memgetfunc_t memory_pool_t::provider

Memory blocks provider for this pool.

7.46 pipe_t Struct Reference

Structure representing a pipe object.

#include <chpipes.h>
Collaboration diagram for `pipe_t`:

Data Fields

- `uint8_t * buffer`
  
  Pointer to the pipe buffer.

- `uint8_t * top`
  
  Pointer to the location after the buffer.

- `uint8_t * wrptr`
  
  Write pointer.

- `uint8_t * rdptr`
  
  Read pointer.

- `size_t cnt`
Bytes in the pipe.

- **bool reset**
  True if in reset state.
- **thread_reference_t wtr**
  Waiting writer.
- **thread_reference_t rtr**
  Waiting reader.
- **mutex_t cmtx**
  Common access mutex.
- **mutex_t wmtx**
  Write access mutex.
- **mutex_t rmtx**
  Read access mutex.

### 7.46.1 Detailed Description

Structure representing a pipe object.

### 7.46.2 Field Documentation

#### 7.46.2.1 buffer

```c
uint8_t* pipe_t::buffer
```

Pointer to the pipe buffer.

#### 7.46.2.2 top

```c
uint8_t* pipe_t::top
```

Pointer to the location after the buffer.
7.46.2.3 wrptr

```
uint8_t* pipe_t::wrptr
```
Write pointer.

7.46.2.4 rdptr

```
uint8_t* pipe_t::rdptr
```
Read pointer.

7.46.2.5 cnt

```
size_t pipe_t::cnt
```
Bytes in the pipe.

7.46.2.6 reset

```
bool pipe_t::reset
```
True if in reset state.

7.46.2.7 wtr

```
thread_reference_t pipe_t::wtr
```
Waiting writer.

7.46.2.8 rtr

```
thread_reference_t pipe_t::rtr
```
Waiting reader.
7.46.2.9 cmtx

mutex_t pipe_t::cmtx

Common access mutex.

7.46.2.10 wmtx

mutex_t pipe_t::wmtx

Write access mutex.

7.46.2.11 rmtx

mutex_t pipe_t::rmtx

Read access mutex.

7.47 pool_header Struct Reference

Memory pool free object header.

#include <chmempools.h>

Collaboration diagram for pool_header:

Data Fields

- struct pool_header * next

  Pointer to the next pool header in the list.
7.47.1 Detailed Description

Memory pool free object header.

7.47.2 Field Documentation

7.47.2.1 next

struct pool_header* pool_header::next

Pointer to the next pool header in the list.

7.48 thread_descriptor_t Struct Reference

Type of a thread descriptor.

#include <chthreads.h>

Collaboration diagram for thread_descriptor_t:
Data Fields

- **const char ∗ name**
  
  Thread name.

- **stkalign_t ∗ wbase**
  
  Pointer to the working area base.

- **stkalign_t ∗ wend**
  
  Pointer to the working area end.

- **tprio_t prio**
  
  Thread priority.

- **func_t funcp**
  
  Thread function pointer.

- **void ∗ arg**
  
  Thread argument.

- **os_instance_t ∗ instance**
  
  OS instance affinity or NULL for current one.

7.48.1 Detailed Description

Type of a thread descriptor.

7.48.2 Field Documentation

7.48.2.1 name

```c
const char* thread_descriptor_t::name
```

Thread name.

7.48.2.2 wbase

```c
stkalign_t* thread_descriptor_t::wbase
```

Pointer to the working area base.

7.48.2.3 wend

```c
stkalign_t* thread_descriptor_t::wend
```

Pointer to the working area end.
7.48.2.4 prio

tprio_t thread_descriptor_t::prio

Thread priority.

7.48.2.5 funcp

tfunc_t thread_descriptor_t::funcp

Thread function pointer.

7.48.2.6 arg

void* thread_descriptor_t::arg

Thread argument.

7.48.2.7 instance

os_instance_t* thread_descriptor_t::instance

OS instance affinity or NULL for current one.

7.49 time_measurement_t Struct Reference

Type of a Time Measurement object.

#include <chtm.h>

Collaboration diagram for time_measurement_t:
Data Fields

- rtcnt_t best
  
  Best measurement.

- rtcnt_t worst
  
  Worst measurement.

- rtcnt_t last
  
  Last measurement.

- ucnt_t n
  
  Number of measurements.

- rttime_t cumulative
  
  Cumulative measurement.

7.49.1 Detailed Description

Type of a Time Measurement object.

Note

The maximum measurable time period depends on the implementation of the realtime counter and its clock frequency.

The measurement is not 100% cycle-accurate, it can be in excess of few cycles depending on the compiler and target architecture.

Interrupts can affect measurement if the measurement is performed with interrupts enabled.

7.49.2 Field Documentation

7.49.2.1 best

\texttt{rtcnt\_t time\_measurement\_t::best}

Best measurement.

7.49.2.2 worst

\texttt{rtcnt\_t time\_measurement\_t::worst}

Worst measurement.
7.49.2.3  last

rtcnt_t time_measurement_t::last

Last measurement.

7.49.2.4  n

ucnt_t time_measurement_t::n

Number of measurements.

7.49.2.5  cumulative

rttime_t time_measurement_t::cumulative

Cumulative measurement.

7.50  tm_calibration_t Struct Reference

Type of a time measurement calibration data.

#include <ctm.h>

Collaboration diagram for tm_calibration_t:

Data Fields

- rtcnt_t offset
  
  Measurement calibration value.
7.50.1 Detailed Description

Type of a time measurement calibration data.

7.50.2 Field Documentation

7.50.2.1 offset

rtcnt_t tm_calibration_t::offset

Measurement calibration value.

7.51 trace_buffer_t Struct Reference

Trace buffer header.

#include <chtrace.h>
Collaboration diagram for trace_buffer_t:

Data Fields

- `uint16_t suspended`
  
  Suspended trace sources mask.

- `uint16_t size`
  
  Trace buffer size (entries).

- `trace_event_t * ptr`
  
  Pointer to the buffer front.
7.51.1 Detailed Description

Trace buffer header.

7.51.2 Field Documentation

7.51.2.1 suspended

```c
uint16_t trace_buffer_t::suspended
```

Suspended trace sources mask.

7.51.2.2 size

```c
uint16_t trace_buffer_t::size
```

Trace buffer size (entries).

7.51.2.3 ptr

```c
trace_event_t* trace_buffer_t::ptr
```

Pointer to the buffer front.

7.51.2.4 buffer

```c
trace_event_t trace_buffer_t::buffer[CH_DBG_TRACE_BUFFER_SIZE]
```

Ring buffer.
7.52  trace_event_t Struct Reference

Trace buffer record.

#include <chtrace.h>

Collaboration diagram for trace_event_t:

```
Data Fields

- uint32_t type:3
  Record type.
- uint32_t state:5
  Switched out thread state.
- uint32_t rtimestamp:24
  Accurate time stamp.
- systime_t time
  System time stamp of the switch event.
- thread_t * ntp
  Switched in thread.
- void * wtobjp
  Object where going to sleep.
```
• struct {
    thread_t *ntp
    Switched in thread.
    void *wtobjp
    Object where going to sleep.
} sw

    Structure representing a context switch.

• thread_t *tp
    Thread made ready.
• msg_t msg
    Ready message.
• struct {
    thread_t *tp
    Thread made ready.
    msg_t msg
    Ready message.
} rdy

    Structure representing a thread becoming ready.

• const char *name
    ISR function name taken using func.
• struct {
    const char *name
    ISR function name taken using func.
} isr

    Structure representing an ISR enter.

• const char *reason
    Halt error string.
• struct {
    const char *reason
    Halt error string.
} halt

    Structure representing an halt.

• void *up1
    Trace user parameter 1.
• void *up2
    Trace user parameter 2.
• struct {
    void *up1
    Trace user parameter 1.
    void *up2
    Trace user parameter 2.
} user

    User trace structure.

7.52.1 Detailed Description

Trace buffer record.
7.52.2 Field Documentation

7.52.2.1 type

uint32_t trace_event_t::type

Record type.

7.52.2.2 state

uint32_t trace_event_t::state

Switched out thread state.

7.52.2.3 rtstamp

uint32_t 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.

7.52.2.4 time

systime_t trace_event_t::time

System time stamp of the switch event.

7.52.2.5 ntp

thread_t* trace_event_t::ntp

Switched in thread.
7.52.2.6  wtobjp

    void* trace_event_t::wtobjp

    Object where going to sleep.

7.52.2.7  sw

    struct { ... } trace_event_t::sw

    Structure representing a context switch.

7.52.2.8  tp

    thread_t* trace_event_t::tp

    Thread made ready.

7.52.2.9  msg

    msg_t trace_event_t::msg

    Ready message.

7.52.2.10  rdy

    struct { ... } trace_event_t::rdy

    Structure representing a thread becoming ready.

7.52.2.11  name

    const char* trace_event_t::name

    ISR function name taken using func.
7.52.2.12 isr

struct { ... } trace_event_t::isr

Structure representing an ISR enter.

7.52.2.13 reason

const char* trace_event_t::reason

Halt error string.

7.52.2.14 halt

struct { ... } trace_event_t::halt

Structure representing a halt.

7.52.2.15 up1

void* trace_event_t::up1

Trace user parameter 1.

7.52.2.16 up2

void* trace_event_t::up2

Trace user parameter 2.

7.52.2.17 user

struct { ... } trace_event_t::user

User trace structure.
Chapter 8

File Documentation

8.1 ch.h File Reference

ChibiOS/RT main include file.

```c
#include "chlicense.h"
#include "chconf.h"
#include "chchecks.h"
#include "chrestrictions.h"
#include "clearly.h"
#include "chrfcu.h"
#include "chdebug.h"
#include "ctime.h"
#include "chlists.h"
#include "chalign.h"
#include "chtrace.h"
#include "chport.h"
#include "chtm.h"
#include "chstats.h"
#include "chobjects.h"
#include "chsys.h"
#include "chinstances.h"
#include "chvt.h"
#include "chschd.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 "7.0.1"
  Kernel version string.
• #define CH_KERNEL_MAJOR 7
  Kernel version major number.
• #define CH_KERNEL_MINOR 0
  Kernel version minor number.
• #define CH_KERNEL_PATCH 1
  Kernel version patch number.

Constants for configuration options

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

8.1.1 Detailed Description

ChibiOS/RT main include file.

8.2 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.

8.2.1 Detailed Description

Memory alignment macros and structures.
8.3 chbsem.h File Reference

Binary semaphores structures and macros.

Data Structures

- struct ch_binary_semaphore
  
  Binary semaphore type.

Macros

- #define __BSEMAPHORE_DATA(name, taken) {__SEMAPHORE_DATA(name.sem, ((taken) ? 0 : 1))}
  
  Data part of a static semaphore initializer.

- #define BSEMAPHORE_DECL(name, taken) binary_semaphore_t name = __BSEMAPHORE_DATA(name, taken)
  
  Static semaphore initializer.

Typedefs

- typedef struct ch_binary_semaphore binary_semaphore_t
  
  Binary semaphore type.

Functions

- static void chBSemObjectInit (binary_semaphore_t *bsp, bool taken)
  
  Initializes a binary semaphore.

- static msg_t chBSemWait (binary_semaphore_t *bsp)
  
  Wait operation on the binary semaphore.

- static msg_t chBSemWaitS (binary_semaphore_t *bsp)
  
  Wait operation on the binary semaphore.

- static msg_t chBSemWaitTimeoutS (binary_semaphore_t *bsp, sysinterval_t timeout)
  
  Wait operation on the binary semaphore.

- static msg_t chBSemWaitTimeout (binary_semaphore_t *bsp, sysinterval_t timeout)
  
  Wait operation on the binary semaphore.

- static void chBSemResetI (binary_semaphore_t *bsp, bool taken)
  
  Reset operation on the binary semaphore.

- static void chBSemReset (binary_semaphore_t *bsp, bool taken)
  
  Reset operation on the binary semaphore.

- static void chBSemSignalI (binary_semaphore_t *bsp)
  
  Performs a signal operation on a binary semaphore.

- static void chBSemSignal (binary_semaphore_t *bsp)
  
  Performs a signal operation on a binary semaphore.

- static bool chBSemGetStateI (const binary_semaphore_t *bsp)
  
  Returns the binary semaphore current state.
8.3.1 Detailed Description

Binary semaphores structures and macros.

Binary semaphores related APIs and services.

**Operation mode**

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

- **Taken**, when its counter has a value of zero or lower than zero. A negative number represent the number of threads queued on the binary semaphore.
- **Not taken**, when its counter has a value of one.

Binary semaphores are different from mutexes because there is no concept of ownership, a binary semaphore can be taken by a thread and signaled by another thread or an interrupt handler, mutexes can only be taken and released by the same thread. Another difference is that binary semaphores, unlike mutexes, do not implement the priority inheritance protocol.

In order to use the binary semaphores APIs the CH_CFG_USE_SEMAPHORES option must be enabled in chconf.h.

8.4 chchecks.h File Reference

Configuration file checks header.

8.4.1 Detailed Description

Configuration file checks header.

8.5 chcond.c File Reference

Condition Variables code.

#include "ch.h"
8.6 chcond.h File Reference

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.

8.5.1 Detailed Description

Condition Variables code.

8.6 chcond.h File Reference

Condition Variables macros and structures.

Data Structures

- struct condition_variable
  condition_variable_t structure.

Macros

- #define __CONDVAR_DATA(name) (__CH_QUEUE_DATA(name.queue))
  Data part of a static condition variable initializer.
- #define CONDVAR_DECL(name) condition_variable_t name = __CONDVAR_DATA(name)
  Static condition variable initializer.

Typedefs

- typedef struct condition_variable condition_variable_t
  condition_variable_t structure.
Functions

- void chCondObjectInit (condition_variable_t *cp)
  Initializes a condition_variable_t structure.
- void chCondSignal (condition_variable_t *cp)
  Signals one thread that is waiting on the condition variable.
- void 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.

8.6.1 Detailed Description

Condition Variables macros and structures.

8.7 chconf.h File Reference

Configuration file template.

Macros

System settings

- #define CH_CFG_SMP_MODE FALSE
  Handling of instances.

System timers settings

- #define CH_CFG_ST_RESOLUTION 32
  System time counter resolution.
- #define CH_CFG_ST_FREQUENCY 10000
  System tick frequency.
- #define CH_CFG_INTERVALS_SIZE 32
  Time intervals data size.
- #define CH_CFG_TIME_TYPES_SIZE 32
  Time types data size.
- #define CH_CFG_ST_TIMEDELTA 2
  Time delta constant for the tick-less mode.
Kernel parameters and options

- `#define CH_CFG_TIME_QUANTUM 0`
  Round robin interval.
- `#define CH_CFG_NO_IDLE_THREAD FALSE`
  Idle thread automatic spawn suppression.

Performance options

- `#define CH_CFG_OPTIMIZE_SPEED TRUE`
  OS optimization.

Subsystem options

- `#define CH_CFG_USE_TM TRUE`
  Time Measurement APIs.
- `#define CH_CFG_USE_TIMESTAMP TRUE`
  Time Stamps APIs.
- `#define CH_CFG_USE_REGISTRY TRUE`
  Threads registry APIs.
- `#define CH_CFG_USE_WAITEXIT TRUE`
  Threads synchronization APIs.
- `#define CH_CFG_USE_SEMAPHORES TRUE`
  Semaphores APIs.
- `#define CH_CFG_USE_SEMAPHORES_PRIORITY FALSE`
  Semaphores queuing mode.
- `#define CH_CFG_USE_MUTEXES TRUE`
  Mutexes APIs.
- `#define CH_CFG_USE_MUTEXES_RECURSIVE FALSE`
  Enables recursive behavior on mutexes.
- `#define CH_CFG_USE_CONDVARS TRUE`
  Conditional Variables APIs.
- `#define CH_CFG_USE_CONDVARS_TIMEOUT TRUE`
  Conditional Variables APIs with timeout.
- `#define CH_CFG_USE_EVENTS TRUE`
  Events Flags APIs.
- `#define CH_CFG_USE_EVENTS_TIMEOUT TRUE`
  Events Flags APIs with timeout.
- `#define CH_CFG_USE_MESSAGES TRUE`
  Synchronous Messages APIs.
- `#define CH_CFG_USE_MESSAGES_PRIORITY FALSE`
  Synchronous Messages queuing mode.
- `#define CH_CFG_USE_DYNAMIC TRUE`
  Dynamic Threads APIs.

OSLIB options

- `#define CH_CFG_USE_MAILBOXES TRUE`
  Mailboxes APIs.
- `#define CH_CFG_USE_MEMCORE TRUE`
  Core Memory Manager APIs.
- `#define CH_CFG_MEMCORE_SIZE 0`
  Managed RAM size.
- `#define CH_CFG_USE_HEAP TRUE`
  Heap Allocator APIs.
- `#define CH_CFG_USE_MEMPOOLS TRUE`
  Memory Pools Allocator APIs.
• #define CH_CFG_USE_OBJ_FIFOS TRUE
  Objects FIFOs APIs.
• #define CH_CFG_USE_PIPES TRUE
  Pipes APIs.
• #define CH_CFG_USE_OBJ_CACHES TRUE
  Objects Caches APIs.
• #define CH_CFG_USE_DELEGATES TRUE
  Delegate threads APIs.
• #define CH_CFG_USE_JOBS TRUE
  Jobs Queues APIs.

Objects factory options

• #define CH_CFG_USE_FACTORY TRUE
  Objects Factory APIs.
• #define CH_CFG_FACTORY_MAX_NAMES_LENGTH 8
  Maximum length for object names.
• #define CH_CFG_FACTORY_OBJECTS_REGISTRY TRUE
  Enables the registry of generic objects.
• #define CH_CFG_FACTORY_GENERIC_BUFFERS TRUE
  Enables factory for generic buffers.
• #define CH_CFG_FACTORY_SEMAPHORES TRUE
  Enables factory for semaphores.
• #define CH_CFG_FACTORY_MAILBOXES TRUE
  Enables factory for mailboxes.
• #define CH_CFG_FACTORY_OBJ_FIFOS TRUE
  Enables factory for objects FIFOs.
• #define CH_CFG_FACTORY_PIPES TRUE
  Enables factory for Pipes.

Debug options

• #define CH_DBG_STATISTICS FALSE
  Debug option, kernel statistics.
• #define CH_DBG_SYSTEM_STATE_CHECK TRUE
  Debug option, system state check.
• #define CH_DBG_ENABLE_CHECKS TRUE
  Debug option, parameters checks.
• #define CH_DBG_ENABLE_ASSERTS TRUE
  Debug option, consistency checks.
• #define CH_DBG_TRACE_MASK CH_DBG_TRACE_MASK_ALL
  Debug option, trace buffer.
• #define CH_DBG_TRACE_BUFFER_SIZE 128
  Trace buffer entries.
• #define CH_DBG_ENABLE_STACK_CHECK TRUE
  Debug option, stack checks.
• #define CH_DBG_FILL_THREADS TRUE
  Debug option, stacks initialization.
• #define CH_DBG_THREADS_PROFILING FALSE
  Debug option, threads profiling.

Kernel hooks

• #define CH_CFG_SYSTEM_EXTRA_FIELDS /* Add system custom fields here.*/
  System structure extension.
• #define CH_CFG_SYSTEM_INIT_HOOK()
  System initialization hook.
8.8 chcustomer.h File Reference

Configuration file template.

A copy of this file must be placed in each project directory, it contains the application specific kernel settings.

8.8 chcustomer.h File Reference

Customer-related info.

Macros

- **#define CH_CUSTOMER_ID_STRING “Santa, North Pole”**
  Customer readable identifier.
- **#define CH_CUSTOMER_ID_CODE “xxxx-yyyy”**
  Customer code.
- **#define CH_CUSTOMER_LICENSE_EOS_DATE 209912**
  End-Of-Support date (yyyyMMdd).
- **#define CH_CUSTOMER_LICENSE_VERSION_YEAR 99**
  Licensed branch year.
- `#define CH_CUSTOMER_LICENSE_VERSION_MONTH 12`
  Licensed branch month.
- `#define CH_LICENSE CH_LICENSE_GPL`
  Current license.
- `#define CH_CUSTOMER_LICENSE_VERSION_DATE`
  Licensed version date in numeric form (yyyymm).

### Licensed Products

- `#define CH_CUSTOMER_LIC_RT TRUE`
- `#define CH_CUSTOMER_LIC_NIL TRUE`
- `#define CH_CUSTOMER_LIC_OSLIB TRUE`
- `#define CH_CUSTOMER_LIC_EX TRUE`
- `#define CH_CUSTOMER_LIC_SB TRUE`
- `#define CH_CUSTOMER_LIC_PORT_CM0 TRUE`
- `#define CH_CUSTOMER_LIC_PORT_CM3 TRUE`
- `#define CH_CUSTOMER_LIC_PORT_CM4 TRUE`
- `#define CH_CUSTOMER_LIC_PORT_CM7 TRUE`
- `#define CH_CUSTOMER_LIC_PORT_CM33 TRUE`
- `#define CH_CUSTOMER_LIC_PORT_ARM79 TRUE`
- `#define CH_CUSTOMER_LIC_PORT_E200Z0 TRUE`
- `#define CH_CUSTOMER_LIC_PORT_E200Z2 TRUE`
- `#define CH_CUSTOMER_LIC_PORT_E200Z3 TRUE`
- `#define CH_CUSTOMER_LIC_PORT_E200Z4 TRUE`

### 8.8.1 Detailed Description

Customer-related info.

### 8.9 chdebug.c File Reference

Debug support code.

```c
#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().*
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.

## 8.9.1 Detailed Description

Debug support code.

## 8.10 chdebug.h File Reference

Debug support macros and structures.

### Data Structures

- struct `ch_system_debug`
  System debug data structure.

### Macros

#### Debug related settings

- `#define CH_DBG_STACK_FILL_VALUE 0x55`
  Fill value for thread stack area in debug mode.

#### Macro Functions

- `#define chDbgCheck(c)`
  Function parameters check.
- `#define chDbgAssert(c, r)`
  Condition assertion.

### Typedefs

- `typedef struct ch_system_debug system_debug_t`
  System debug data structure.

### Functions

- `static void __dbg_object_init (system_debug_t *sdp)`
  Debug support initialization.
8.10.1 Detailed Description

Debug support macros and structures.

8.11 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.

8.11.1 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.
Delegate threads macros and structures.

```c
#include <stdarg.h>
```

## Typedefs

- **typedef msg_t(** delegate_veneer_t **) (va_list *argsp)**
  Type of a delegate veneer function.
- **typedef msg_t(** delegate_fn0_t **) (void)**
  Type of a delegate function with no parameters.
- **typedef msg_t(** delegate_fn1_t **) (msg_t p1)**
  Type of a delegate function with one parameter.
- **typedef msg_t(** delegate_fn2_t **) (msg_t p1, msg_t p2)**
  Type of a delegate function with two parameters.
- **typedef msg_t(** delegate_fn3_t **) (msg_t p1, msg_t p2, msg_t p3)**
  Type of a delegate function with three parameters.
- **typedef msg_t(** delegate_fn4_t **) (msg_t p1, msg_t p2, msg_t p3, msg_t p4)**
  Type of a delegate function with four parameters.

## Functions

- **msg_t __ch_delegate_fn0 (va_list *argsp)**
  Veneer for functions with no parameters.
- **msg_t __ch_delegate_fn1 (va_list *argsp)**
  Veneer for functions with one parameter.
- **msg_t __ch_delegate_fn2 (va_list *argsp)**
  Veneer for functions with two parameters.
- **msg_t __ch_delegate_fn3 (va_list *argsp)**
  Veneer for functions with three parameters.
- **msg_t __ch_delegate_fn4 (va_list *argsp)**
  Veneer for functions with four parameters.
- **void chDelegateDispatch (void)**
  Call messages dispatching.
- **msg_t chDelegateDispatchTimeout (sysinterval_t timeout)**
  Call messages dispatching with timeout.
- **msg_t chDelegateCallVeneer (thread_t *tp, delegate_veneer_t veneer,...)**
  Triggers a function call on a delegate thread.
- **static msg_t chDelegateCallDirect0 (thread_t *tp, delegate_fn0_t func)**
  Direct call to a function with no parameters.
- **static msg_t chDelegateCallDirect1 (thread_t *tp, delegate_fn1_t func, msg_t p1)**
  Direct call to a function with one parameter.
- **static msg_t chDelegateCallDirect2 (thread_t *tp, delegate_fn2_t func, msg_t p1, msg_t p2)**
  Direct call to a function with two parameters.
- **static msg_t chDelegateCallDirect3 (thread_t *tp, delegate_fn3_t func, msg_t p1, msg_t p2, msg_t p3)**
  Direct call to a function with three parameters.
- **static msg_t chDelegateCallDirect4 (thread_t *tp, delegate_fn4_t func, msg_t p1, msg_t p2, msg_t p3, msg_t p4)**
  Direct call to a function with four parameters.
8.12.1 Detailed Description

Delegate threads macros and structures.

8.13 chdynamic.c File Reference

Dynamic threads code.

```c
#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.

8.13.1 Detailed Description

Dynamic threads code.

8.14 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.

8.14.1 Detailed Description

Dynamic threads macros and structures.
8.15  chearly.h File Reference

Early forward types declarations header.

#include "chtypes.h"

Macros

- `#define __CH_STRINGIFY(a) #a`  
  Utility to make the parameter a quoted string.
- `#define __CH_OFFSETOF(st, m)`  
  Structure field offset utility.
- `#define __CH_USED(x) (void)(x)`  
  Marks an expression result as used.
- `#define likely(x) PORT_LIKELY(x)`  
  Marks a boolean expression as likely true.
- `#define unlikely(x) PORT_UNLIKELY(x)`  
  Marks a boolean expression as likely false.
- `#define threadref(p) ((thread_t *)((void *) (p)))`  
  Safe cast of a queue pointer to a thread pointer.

Typedefs

- `typedef unsigned core_id_t`  
  Type of a core identifier.
- `typedef struct ch_thread thread_t`  
  Type of a thread structure.
- `typedef struct ch_os_instance os_instance_t`  
  Type of an OS instance structure.

Kernel types

- `typedef port_rtcnt_t rtcnt_t`  
- `typedef port_rttime_t rttime_t`  
- `typedef port_syssts_t syssts_t`  
- `typedef port_stkalign_t stkalign_t`  
- `typedef uint8_t tmode_t`  
- `typedef uint8_t tstate_t`  
- `typedef uint8_t trefs_t`  
- `typedef uint8_t tslices_t`  
- `typedef uint32_t tprio_t`  
- `typedef int32_t msg_t`  
- `typedef int32_t eventid_t`  
- `typedef uint32_t eventmask_t`  
- `typedef uint32_t eventflags_t`  
- `typedef int32_t cnt_t`  
- `typedef uint32_t ucnt_t`

Functions

- `void chSysHalt (const char *reason)`  
  Halts the system.
8.15.1 Detailed Description

Early forward types declarations header.

8.16 chevents.c File Reference

Events code.
#include "ch.h"

Functions

- void chEvtRegisterMaskWithFlagsI (event_source_t *esp, event_listener_t *elp, eventmask_t events, eventflags_t wflags)
  Registers an Event Listener on an Event Source.
- void chEvtRegisterMaskWithFlags (event_source_t *esp, event_listener_t *elp, eventmask_t events, eventflags_t wflags)
  Registers an Event Listener on an Event Source.
- void chEvtUnregister (event_source_t *esp, event_listener_t *elp)
  Unregisters an Event Listener from its Event Source.
- eventmask_t chEvtGetAndClearEventsI (eventmask_t events)
  Clears the pending events specified in the events mask.
- eventmask_t chEvtGetAndClearEvents (eventmask_t events)
  Clears the pending events specified in the events mask.
- eventmask_t chEvtAddEvents (eventmask_t events)
  Adds (OR) a set of events to the current thread, this is much faster than using chEvtBroadcast() or chEvtSignal().
- eventflags_t chEvtGetAndClearFlagsI (event_listener_t *elp)
  Returns the unmasked flags associated to an event_listener_t.
- eventflags_t chEvtGetAndClearFlags (event_listener_t *elp)
  Returns the flags associated to an event_listener_t.
- void chEvtSignalI (thread_t *tp, eventmask_t events)
  Adds a set of event flags directly to the specified thread_t.
- void chEvtSignal (thread_t *tp, eventmask_t events)
  Adds a set of event flags directly to the specified thread_t.
- void chEvtBroadcastFlagsI (event_source_t *esp, eventflags_t flags)
  Signals all the Event Listeners registered on the specified Event Source.
- void chEvtBroadcastFlags (event_source_t *esp, eventflags_t flags)
  Signals all the Event Listeners registered on the specified Event Source.
- void chEvtDispatch (const evhandler_t *handlers, eventmask_t events)
  Invokes the event handlers associated to an event flags mask.
- eventmask_t chEvtWaitOne (eventmask_t events)
  Waits for exactly one of the specified events.
- eventmask_t chEvtWaitAny (eventmask_t events)
  Waits for any of the specified events.
- eventmask_t chEvtWaitAll (eventmask_t events)
  Waits for all the specified events.
- eventmask_t chEvtWaitOneTimeout (eventmask_t events, sysinterval_t timeout)
  Waits for exactly one of the specified events.
- eventmask_t chEvtWaitAnyTimeout (eventmask_t events, sysinterval_t timeout)
  Waits for any of the specified events.
- eventmask_t chEvtWaitAllTimeout (eventmask_t events, sysinterval_t timeout)
  Waits for all the specified events.
8.16.1 Detailed Description

Events code.

8.17 chevents.h File Reference

Events macros and structures.

Data Structures

- `struct event_listener`
  Event Listener structure.
- `struct event_source`
  Event Source structure.

Macros

- `#define ALL_EVENTS ((eventmask_t)-1)`
  All events allowed mask.
- `#define EVENT_MASK(eid) ((eventmask_t)(1 << (eventmask_t)(eid)))`
  Returns an event mask from an event identifier.
- `#define __EVENTSOURCE_DATA(name) {(event_listener_t *)(&name)}`
  Data part of a static event source initializer.
- `#define EVENTSOURCE_DECL(name) event_source_t name = __EVENTSOURCE_DATA(name)`
  Static event source initializer.

Typedefs

- `typedef struct event_source event_source_t`
  Event Source structure.
- `typedef void(∗evhandler_t) (eventid_t id)`
  Event Handler callback function.

Functions

- `void chEvtRegisterMaskWithFlagsI (event_source_t ∗esp, event_listener_t ∗elp, eventmask_t events, eventflags_t wflags)`
  Registers an Event Listener on an Event Source.
- `void chEvtRegisterMaskWithFlags (event_source_t ∗esp, event_listener_t ∗elp, eventmask_t events, eventflags_t wflags)`
  Registers an Event Listener on an Event Source.
- `void chEvtUnregister (event_source_t ∗esp, event_listener_t ∗elp)`
  Unregisters an Event Listener from its Event Source.
- `eventmask_t chEvtGetAndClearEventsI (eventmask_t events)`
  Clears the pending events specified in the events mask.
- `eventmask_t chEvtGetAndClearEvents (eventmask_t events)`
  Clears the pending events specified in the events mask.
Clears the pending events specified in the events mask.

- `eventmask_t chEvtAddEvents (eventmask_t events)`
  
  Adds (OR) a set of events to the current thread, this is much faster than using `chEvtBroadcast()` or `chEvtSignal()`.

- `eventflags_t chEvtGetAndClearFlagsI (event_listener_t *elp)`
  
  Returns the unmasked flags associated to an event_listener_t.

- `eventflags_t chEvtGetAndClearFlags (event_listener_t *elp)`
  
  Returns the flags associated to an event_listener_t.

- `void chEvtSignal (thread_t *tp, eventmask_t events)`
  
  Adds a set of event flags directly to the specified thread_t.

- `void 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 chEvtBroadcastII (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.

### 8.17.1 Detailed Description

Events macros and structures.
ChibiOS objects factory and registry code.

#include <string.h>
#include "ch.h"

Functions

* void __factory_init (void)
  Initializes the objects factory.
* registered_object_t * chFactoryRegisterObject (const char *name, void *objp)
  Registers a generic object.
* registered_object_t * chFactoryFindObject (const char *name)
  Retrieves a registered object.
* registered_object_t * chFactoryFindObjectByPointer (void *objp)
  Retrieves a registered object by pointer.
* void chFactoryReleaseObject (registered_object_t *rop)
  Releases a registered object.
* dyn_buffer_t * chFactoryCreateBuffer (const char *name, size_t size)
  Creates a generic dynamic buffer object.
* dyn_buffer_t * chFactoryFindBuffer (const char *name)
  Retrieves a dynamic buffer object.
* void chFactoryReleaseBuffer (dyn_buffer_t *dbp)
  Releases a dynamic buffer object.
* dyn_semaphore_t * chFactoryCreateSemaphore (const char *name, cnt_t n)
  Creates a dynamic semaphore object.
* dyn_semaphore_t * chFactoryFindSemaphore (const char *name)
  Retrieves a dynamic semaphore object.
* void chFactoryReleaseSemaphore (dyn_semaphore_t *dsp)
  Releases a dynamic semaphore object.
* dyn_mailbox_t * chFactoryCreateMailbox (const char *name, size_t n)
  Creates a dynamic mailbox object.
* dyn_mailbox_t * chFactoryFindMailbox (const char *name)
  Retrieves a dynamic mailbox object.
* void chFactoryReleaseMailbox (dyn_mailbox_t *dmp)
  Releases a dynamic mailbox object.
* dyn_objects_fifo_t * chFactoryCreateObjectsFIFO (const char *name, size_t objsize, size_t objn, unsigned objalign)
  Creates a dynamic "objects FIFO" object.
* dyn_objects_fifo_t * chFactoryFindObjectsFIFO (const char *name)
  Retrieves a dynamic "objects FIFO" object.
* void chFactoryReleaseObjectsFIFO (dyn_objects_fifo_t *dofp)
  Releases a dynamic "objects FIFO" object.
* dyn_pipe_t * chFactoryCreatePipe (const char *name, size_t size)
  Creates a dynamic pipe object.
* dyn_pipe_t * chFactoryFindPipe (const char *name)
  Retrieves a dynamic pipe object.
* void chFactoryReleasePipe (dyn_pipe_t *dpp)
  Releases a dynamic pipe object.
Variables

- **objects_factory_t ch_factory**
  Factory object static instance.

8.18.1 Detailed Description

ChibiOS objects factory and registry code.

8.19 chfactory.h File Reference

ChibiOS objects factory structures and macros.

Data Structures

- **struct ch_dyn_element**
  Type of a dynamic object list element.
- **struct ch_dyn_list**
  Type of a dynamic object list.
- **struct ch_registered_static_object**
  Type of a registered object.
- **struct ch_dyn_object**
  Type of a dynamic buffer object.
- **struct ch_dyn_semaphore**
  Type of a dynamic semaphore.
- **struct ch_dyn_mailbox**
  Type of a dynamic buffer object.
- **struct ch_dyn_objects_fifo**
  Type of a dynamic buffer object.
- **struct ch_dyn_pipe**
  Type of a dynamic pipe object.
- **struct ch_objects_factory**
  Type of the factory main object.

Macros

- **#define CH_CFG_FACTORY_MAX_NAMES_LENGTH 8**
  Maximum length for object names.
- **#define CH_CFG_FACTORY_OBJECTS_REGISTRY TRUE**
  Enables the registry of generic objects.
- **#define CH_CFG_FACTORY_GENERIC_BUFFERS TRUE**
  Enables factory for generic buffers.
- **#define CH_CFG_FACTORY_SEMAPHORES TRUE**
  Enables factory for semaphores.
- **#define CH_CFG_FACTORY.Mailboxes TRUE**
  Enables factory for mailboxes.
- **#define CH_CFG_FACTORY_OBJ_FIFOs TRUE**
Enables factory for objects FIFOs.
• #define CH_CFG_FACTORY_OBJ_FIFOS TRUE
  Enables factory for objects FIFOs.
• #define CH_CFG_FACTORY_PIPES TRUE
  Enables factory for pipes.
• #define CH_CFG_FACTORY_SEMAPHORES FALSE
  Enables factory for semaphores.
• #define CH_CFG_FACTORY_MAILBOXES FALSE
  Enables factory for mailboxes.
• #define CH_CFG_FACTORY_OBJ_FIFOS FALSE
  Enables factory for objects FIFOs.
• #define CH_CFG_FACTORY_PIPES FALSE
  Enables factory for pipes.

**Typedefs**

• typedef struct ch_dyn_element dyn_element_t
  Type of a dynamic object list element.
• typedef struct ch_dyn_list dyn_list_t
  Type of a dynamic object list.
• typedef struct ch_registered_static_object registered_object_t
  Type of a registered object.
• typedef struct ch_dyn_object dyn_buffer_t
  Type of a dynamic buffer object.
• typedef struct ch_dyn_semaphore dyn_semaphore_t
  Type of a dynamic semaphore.
• typedef struct ch_dyn_mailbox dyn_mailbox_t
  Type of a dynamic buffer object.
• typedef struct ch_dyn_objects_fifo dyn_objects_fifo_t
  Type of a dynamic buffer object.
• typedef struct ch_dyn_pipe dyn_pipe_t
  Type of a dynamic pipe object.
• typedef struct ch_objects_factory objects_factory_t
  Type of the factory main object.

**Functions**

• void __factory_init (void)
  Initializes the objects factory.
• registered_object_t * chFactoryRegisterObject (const char *name, void *objp)
  Registers a generic object.
• registered_object_t * chFactoryFindObject (const char *name)
  Retrieves a registered object.
• registered_object_t * chFactoryFindObjectByPointer (void *objp)
  Retrieves a registered object by pointer.
• void chFactoryReleaseObject (registered_object_t *rop)
  Releases a registered object.
• dyn_buffer_t * chFactoryCreateBuffer (const char *name, size_t size)
  Creates a generic dynamic buffer object.
• `dyn_buffer_t * chFactoryFindBuffer (const char *name)`
  Retrieves a dynamic buffer object.

• `void chFactoryReleaseBuffer (dyn_buffer_t *dbp)`
  Releases a dynamic buffer object.

• `dyn_semaphore_t * chFactoryCreateSemaphore (const char *name)`
  Creates 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.

• `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.

• `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.

• `void chFactoryReleasePipe (dyn_pipe_t *dpp)`
  Releases a dynamic pipe object.

• `static dyn_element_t * chFactoryDuplicateReference (dyn_element_t *dep)`
  Duplicates an object reference.

• `static void * chFactoryGetObject (registered_object_t *rop)`
  Returns the pointer to the inner registered object.

• `static size_t chFactoryGetBufferSize (dyn_buffer_t *dbp)`
  Returns the size of a generic dynamic buffer object.

• `static uint8_t * chFactoryGetBuffer (dyn_buffer_t *dbp)`
  Returns the pointer to the inner buffer.

• `static semaphore_t * chFactoryGetSemaphore (dyn_semaphore_t *dsp)`
  Returns the pointer to the inner semaphore.

• `static mailbox_t * chFactoryGetMailbox (dyn_mailbox_t *dmp)`
  Returns the pointer to the inner mailbox.

• `static objects_fifo_t * chFactoryGetObjectsFIFO (dyn_objects_fifo_t *dofp)`
  Returns the pointer to the inner objects FIFO.

• `static pipe_t * chFactoryGetPipe (dyn_pipe_t *dpp)`
  Returns the pointer to the inner pipe.

### 8.19.1 Detailed Description

ChibiOS objects factory structures and macros.
8.20 chinstances.c File Reference

OS instances code.

#include "ch.h"

Functions

• static void __idle_thread (void *p)
  
  This function implements the idle thread infinite loop.

• void chInstanceObjectInit (os_instance_t *oip, const os_instance_config_t *oicp)
  
  Initializes a system instance.

8.20.1 Detailed Description

OS instances code.

8.21 chinstances.h File Reference

OS instances macros and structures.

Macros

• #define __instance_get_curthread(oip) (oip)->rlist.current
  
  Current thread pointer get macro.

• #define __instance_set_curthread(oip, tp) (oip)->rlist.current = (tp)
  
  Current thread pointer set macro.

Functions

• void chInstanceObjectInit (os_instance_t *oip, const os_instance_config_t *oicp)
  
  Initializes a system instance.

8.21.1 Detailed Description

OS instances macros and structures.

8.22 chjobs.h File Reference

Jobs Queues structures and macros.
Data Structures

- struct ch_jobs_queue
  Type of a jobs queue.
- struct ch_job_descriptor
  Type of a job descriptor.

Macros

- #define MSG_JOB_NULL ((msg_t)-2)
  Dispatcher return code in case of a JOB_NULL has been received.

Typedefs

- typedef struct ch_jobs_queue jobs_queue_t
  Type of a jobs queue.
- typedef void(job_function_t) (void *arg)
  Type of a job function.
- typedef struct ch_job_descriptor job_descriptor_t
  Type of a job descriptor.

Functions

- static void chJobObjectInit (jobs_queue_t *jqp, size_t jobsn, job_descriptor_t *jobsbuf, msg_t *msgbuf)
  Initializes a jobs queue object.
- static job_descriptor_t * chJobGet (jobs_queue_t *jqp)
  Allocates a free job object.
- static job_descriptor_t * chJobGetI (jobs_queue_t *jqp)
  Allocates a free job object.
- static job_descriptor_t * chJobGetTimeoutS (jobs_queue_t *jqp, sysinterval_t timeout)
  Allocates a free job object.
- static job_descriptor_t * chJobGetTimeout (jobs_queue_t *jqp, sysinterval_t timeout)
  Allocates a free job object.
- static void chJobPostI (jobs_queue_t *jqp, job_descriptor_t *jp)
  Posts a job object.
- static void chJobPostS (jobs_queue_t *jqp, job_descriptor_t *jp)
  Posts a job object.
- static void chJobPost (jobs_queue_t *jqp, job_descriptor_t *jp)
  Posts a job object.
- static void chJobPostAheadI (jobs_queue_t *jqp, job_descriptor_t *jp)
  Posts an high priority job object.
- static void chJobPostAheadS (jobs_queue_t *jqp, job_descriptor_t *jp)
  Posts an high priority job object.
- static void chJobPostAhead (jobs_queue_t *jqp, job_descriptor_t *jp)
  Posts an high priority job object.
- static msg_t chJobDispatch (jobs_queue_t *jqp)
  Waits for a job then executes it.
- static msg_t chJobDispatchTimeout (jobs_queue_t *jqp, sysinterval_t timeout)
  Waits for a job then executes it.
8.22.1 Detailed Description

Jobs Queues structures and macros.

This module implements queues of generic jobs to be delegated asynchronously to a pool of dedicated threads. Operations defined for Jobs Queues

- Get: An job object is taken from the pool of the available jobs.
- Post: A job is posted to the queue, it will be returned to the pool after execution.

8.23 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 "1.3.0"`  
  OS Library version string.
- `#define CH_OSLIB_MAJOR 1`  
  OS Library version major number.
- `#define CH_OSLIB_MINOR 3`  
  OS Library version minor number.
- `#define CH_OSLIB_PATCH 0`  
  OS Library version patch number.

Functions

- `static void __oslib_init (void)`  
  Initialization of all library modules.
8.23.1 Detailed Description

ChibiOS/LIB main include file.

This header includes all the required library headers. This file is meant to be included by `ch.h` not directly by user.

8.24 chlicense.h File Reference

License Module macros and structures.

```c
#include "chversion.h"
#include "chcustomer.h"
```

Macros

- `#define CH_LICENSE_TYPE_STRING "GNU General Public License 3 (GPL3)"
  License identification string.`
- `#define CH_LICENSE_ID_STRING "N/A"
  Customer identification string.`
- `#define CH_LICENSE_ID_CODE "N/A"
  Customer code.`
- `#define CH_LICENSE_MODIFIABLE_CODE TRUE
  Code modifiability restrictions.`
- `#define CH_LICENSE_FEATURES CH_FEATURES_FULL
  Code functionality restrictions.`
- `#define CH_LICENSE_MAX_DEPLOY CH_DEPLOY_UNLIMITED
  Code deploy restrictions.`

Allowed Features Levels

- `#define CH_FEATURES_BASIC 0`
- `#define CH_FEATURES_INTERMEDIATE 1`
- `#define CH_FEATURES_FULL 2`

Deployment Options

- `#define CH_DEPLOY_UNLIMITED -1`
- `#define CH_DEPLOY_NONE 0`

Licensing Options

- `#define CH_LICENSE_GPL 0`
- `#define CH_LICENSE_GPL_EXCEPTION 1`
- `#define CH_LICENSE_COMMERCIAL_FREE 2`
- `#define CH_LICENSE_COMMERCIAL_DEV_1000 3`
- `#define CH_LICENSE_COMMERCIAL_DEV_5000 4`
- `#define CH_LICENSE_COMMERCIAL_FULL 5`
- `#define CH_LICENSE_COMMERCIAL_RUNTIME 6`
- `#define CH_LICENSE_PARTNER 7`
8.24.1 Detailed Description

License Module macros and structures.

8.25 chlists.h File Reference

Lists and Queues header.

Data Structures

- struct ch_list
  
  Structure representing a generic single link list header and element.

- struct ch_queue
  
  Structure representing a generic bidirectional linked list header and element.

- struct ch_priority_queue
  
  Structure representing a generic priority-ordered bidirectional linked list header and element.

- struct ch_delta_list
  
  Delta list element and header structure.

Macros

- #define __CH_QUEUE_DATA(name) {(ch_queue_t *)&name, (ch_queue_t *)&name}
  
  Data part of a static queue object initializer.

- #define CH_QUEUE_DECL(name) ch_queue_t name = __CH_QUEUE_DATA(name)
  
  Static queue object initializer.

Typedefs

- typedef struct ch_list ch_list_t
  
  Type of a generic single link list header and element.

- typedef struct ch_queue ch_queue_t
  
  Type of a generic bidirectional linked list header and element.

- typedef struct ch_priority_queue ch_priority_queue_t
  
  Type of a generic priority-ordered bidirectional linked list header and element.

- typedef struct ch_delta_list ch_delta_list_t
  
  Type of a generic bidirectional linked delta list header and element.
Functions

- static void ch_list_init (ch_list_t *lp)
  
  List initialization.
- static bool ch_list_isempty (ch_list_t *lp)
  
  Evaluates to true if the specified list is empty.
- static bool ch_list_notempty (ch_list_t *lp)
  
  Evaluates to true if the specified list is not empty.
- static void ch_list_link (ch_list_t *lp, ch_list_t *p)
  
  Pushes an element on top of a stack list.
- static void ch_list_unlink (ch_list_t *lp)
  
  Pops an element from the top of a stack list and returns it.
- static void ch_queue_init (ch_queue_t *qp)
  
  Queue initialization.
- static bool ch_queue_isempty (const ch_queue_t *qp)
  
  Evaluates to true if the specified queue is empty.
- static bool ch_queue_notempty (const ch_queue_t *qp)
  
  Evaluates to true if the specified queue is not empty.
- static void ch_queue_insert (ch_queue_t *qp, ch_queue_t *p)
  
  Inserts an element into a queue.
- static ch_queue_t * ch_queue_fifo_remove (ch_queue_t *qp)
  
  Removes the first-out element from a queue and returns it.
- static ch_queue_t * ch_queue_lifo_remove (ch_queue_t *qp)
  
  Removes the last-out element from a queue and returns it.
- static ch_queue_t * ch_queue_dequeue (ch_queue_t *p)
  
  Removes an element from a queue and returns it.
- static void ch_pqueue_init (ch_priority_queue_t *pqp)
  
  Priority queue initialization.
- static ch_priority_queue_t * ch_pqueue_remove_highest (ch_priority_queue_t *pqp)
  
  Removes the highest priority element from a priority queue and returns it.
- static ch_priority_queue_t * ch_pqueue_insert_behind (ch_priority_queue_t *pqp, ch_priority_queue_t *p)
  
  Inserts an element in the priority queue placing it behind its peers.
- static ch_priority_queue_t * ch_pqueue_insert_ahead (ch_priority_queue_t *pqp, ch_priority_queue_t *p)
  
  Inserts an element in the priority queue placing it ahead of its peers.
- static void ch_dlist_init (ch_delta_list_t *dlhp)
  
  Delta list initialization.
- static bool ch_dlist_isempty (ch_delta_list_t *dlhp)
  
  Evaluates to true if the specified delta list is empty.
- static bool ch_dlist_notempty (ch_delta_list_t *dlhp)
  
  Evaluates to true if the specified delta list is not empty.
- static bool ch_dlist_islast (ch_delta_list_t *dlhp, ch_delta_list_t *dlp)
  
  Last element in the delta list check.
- static bool ch_dlist isFirst (ch_delta_list_t *dlhp, ch_delta_list_t *dlp)
  
  First element in the delta list check.
- static void ch_dlist_insert_after (ch_delta_list_t *dlhp, ch_delta_list_t *dlp, sysinterval_t delta)
  
  Inserts an element after another header element.
- static void ch_dlist_insert_before (ch_delta_list_t *dlhp, ch_delta_list_t *dlp, sysinterval_t delta)
  
  Inserts an element before another header element.
- static void ch_dlist_insert (ch_delta_list_t *dlhp, ch_delta_list_t *dlep, sysinterval_t delta)
  
  Inserts an element in a delta list.
- static ch_delta_list_t * ch_dlist_remove_first (ch_delta_list_t *dlhp)
  
  Dequeues an element from the delta list.
- static ch_delta_list_t * ch_dlist_dequeue (ch_delta_list_t *dlp)
  
  Dequeues an element from the delta list.
8.25.1 Detailed Description

Lists and Queues header.

8.26 chmboxes.c File Reference

Mailboxes code.

#include "ch.h"

Functions

- void chMBObjectInit (mailbox_t *mbp, msg_t *buf, size_t n)
  
  Initializes a mailbox_t object.

- void chMBReset (mailbox_t *mbp)
  
  Resets a mailbox_t object.

- void chMBResetI (mailbox_t *mbp)
  
  Resets a mailbox_t object.

- msg_t chMBPostTimeout (mailbox_t *mbp, msg_t msg, sysinterval_t timeout)
  
  Posts a message into a mailbox.

- msg_t chMBPostTimeoutS (mailbox_t *mbp, msg_t msg, sysinterval_t timeout)
  
  Posts a message into a mailbox.

- msg_t chMBPostI (mailbox_t *mbp, msg_t msg)
  
  Posts a message into a mailbox.

- msg_t chMBPostAheadTimeout (mailbox_t *mbp, msg_t msg, sysinterval_t timeout)
  
  Posts an high priority message into a mailbox.

- msg_t chMBPostAheadTimeoutS (mailbox_t *mbp, msg_t msg, sysinterval_t timeout)
  
  Posts an high priority message into a mailbox.

- msg_t chMBPostAheadI (mailbox_t *mbp, msg_t msg)
  
  Posts an high priority message into a mailbox.

- msg_t chMBFetchTimeout (mailbox_t *mbp, msg_t *msgp, sysinterval_t timeout)
  
  Retrieves a message from a mailbox.

- msg_t chMBFetchTimeoutS (mailbox_t *mbp, msg_t *msgp, sysinterval_t timeout)
  
  Retrieves a message from a mailbox.

- msg_t chMBFetchI (mailbox_t *mbp, msg_t *msgp)
  
  Retrieves a message from a mailbox.

8.26.1 Detailed Description

Mailboxes code.

8.27 chmboxes.h File Reference

Mailboxes macros and structures.
Data Structures

- struct mailbox_t
  Structure representing a mailbox object.

Macros

- #define __MAILBOX_DATA(name, buffer, size)
  Data part of a static mailbox initializer.
- #define MAILBOX_DECL(name, buffer, size) mailbox_t name = __MAILBOX_DATA(name, buffer, size)
  Static mailbox initializer.

Functions

- void chMObjectInit (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.
8.27.1 Detailed Description

Mailboxes macros and structures.

8.28 chmemcore.c File Reference

Core memory manager code.

#include "ch.h"

Functions

• void __core_init (void)
  Low level memory manager initialization.

• void * chCoreAllocFromBaseI (size_t size, unsigned align, size_t offset)
  Allocates a memory block starting from the lowest address upward.

• void * chCoreAllocFromTopI (size_t size, unsigned align, size_t offset)
  Allocates a memory block starting from the top address downward.

• void * chCoreAllocFromBase (size_t size, unsigned align, size_t offset)
  Allocates a memory block starting from the lowest address upward.

• void * chCoreAllocFromTop (size_t size, unsigned align, size_t offset)
  Allocates a memory block starting from the top address downward.

• size_t chCoreGetStatusX (void)
  Core memory status.

Variables

• memcore_t ch_memcore
  Memory core descriptor.

8.28.1 Detailed Description

Core memory manager code.

8.29 chmemcore.h File Reference

Core memory manager macros and structures.

Data Structures

• struct memcore_t
  Type of memory core object.
Macros

- `#define CH_CFG_MEMCORE_SIZE 0`
  Managed RAM size.
- `#define 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 ∗ chCoreAlloc (size_t size)`
  Allocates a memory block.
- `static void ∗ chCoreAlloc (size_t size)`
  Allocates a memory block.

8.29.1 Detailed Description

Core memory manager macros and structures.

8.30 chmemheaps.c File Reference

Memory heaps code.

#include "ch.h"
Functions

- void __heap_init (void)
  Initializes the default heap.
- void chHeapObjectInit (memory_heap_t *heapp, void *buf, size_t size)
  Initializes a memory heap from a static memory area.
- void * chHeapAllocAligned (memory_heap_t *heapp, size_t size, unsigned align)
  Allocates a block of memory from the heap by using the first-fit algorithm.
- void chHeapFree (void *p)
  Frees a previously allocated memory block.
- size_t chHeapStatus (memory_heap_t *heapp, size_t *totalp, size_t *largestp)
  Reports the heap status.

Variables

- static memory_heap_t default_heap
  Default heap descriptor.

8.30.1 Detailed Description

Memory heaps code.

8.31 chmemheaps.h File Reference

Memory heaps macros and structures.

Data Structures

- union heap_header
  Memory heap block header.
- struct memory_heap
  Structure describing a memory heap.

Macros

- #define CH_HEAP_ALIGNMENT 8U
  Minimum alignment used for heap.
- #define CH_HEAP_AREA(name, size)
  Allocation of an aligned static heap buffer.

Typedefs

- typedef struct memory_heap memory_heap_t
  Type of a memory heap.
- typedef union heap_header heap_header_t
  Type of a memory heap header.
### Functions

- `void __heap_init (void)`
  Initializes the default heap.
- `void chHeapObjectInit (memory_heap_t *heap, void *buf, size_t size)`
  Initializes a memory heap from a static memory area.
- `void * chHeapAllocAligned (memory_heap_t *heap, 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 *heap, size_t *totalp, size_t *largestp)`
  Reports the heap status.
- `static void * chHeapAlloc (memory_heap_t *heap, size_t size)`
  Allocates a block of memory from the heap by using the first-fit algorithm.
- `static size_t chHeapGetSize (const void *p)`
  Returns the size of an allocated block.

#### 8.31.1 Detailed Description

Memory heaps macros and structures.

#### 8.32 chmempools.c File Reference

Memory Pools code.

```c
#include "ch.h"
```

### Functions

- `void chPoolObjectInitAligned (memory_pool_t *mp, size_t size, unsigned align, memgetfunc_t provider)`
  Initializes an empty memory pool.
- `void chPoolLoadArray (memory_pool_t *mp, void *p, size_t n)`
  Loads a memory pool with an array of static objects.
- `void * chPoolAllocI (memory_pool_t *mp)`
  Allocates an object from a memory pool.
- `void * chPoolAlloc (memory_pool_t *mp)`
  Allocates an object from a memory pool.
- `void chPoolFreeI (memory_pool_t *mp, void *objp)`
  Releases an object into a memory pool.
- `void chPoolFree (memory_pool_t *mp, void *objp)`
  Releases an object into a memory pool.
- `void chGuardedPoolObjectInitAligned (guarded_memory_pool_t *gmp, size_t size, unsigned align)`
  Initializes an empty guarded memory pool.
- `void chGuardedPoolLoadArray (guarded_memory_pool_t *gmp, void *p, size_t n)`
  Loads a guarded memory pool with an array of static objects.
- `void * chGuardedPoolAllocTimeoutS (guarded_memory_pool_t *gmp, sysinterval_t timeout)`
  Allocates an object from a guarded memory pool.
- `void * chGuardedPoolAllocTimeout (guarded_memory_pool_t *gmp, sysinterval_t timeout)`
  Allocates an object from a guarded memory pool.
- `void chGuardedPoolFree (guarded_memory_pool_t *gmp, void *objp)`
  Releases an object into a guarded memory pool.
8.32.1 Detailed Description

Memory Pools code.

8.33 chmempools.h File Reference

Memory Pools macros and structures.

Data Structures

- struct pool_header
  Memory pool free object header.
- struct memory_pool_t
  Memory pool descriptor.
- struct guarded_memory_pool_t
  Guarded memory pool descriptor.

Macros

- #define __MEMORYPOOL_DATA(name, size, align, provider) {NULL, size, align, provider}
  Data part of a static memory pool initializer.
- #define MEMORYPOOL_DECL(name, size, align, provider) memory_pool_t name = __MEMORYPOOL_DATA(name, size, align, provider)
  Static memory pool initializer.
- #define __GUARDEDMEMORYPOOL_DATA(name, size, align)
  Data part of a static guarded memory pool initializer.
- #define GUARDEDMEMORYPOOL_DECL(name, size, align) guarded_memory_pool_t name = __GUARDEDMEMORYPOOL_DATA(name, size, align)
  Static guarded memory pool initializer.

Functions

- void chPoolObjectInitAligned (memory_pool_t *mp, size_t size, unsigned align, memgetfunc_t provider)
  Initializes an empty memory pool.
- void chPoolLoadArray (memory_pool_t *mp, void *p, size_t n)
  Loads a memory pool with an array of static objects.
- void * chPoolAlloc (memory_pool_t *mp)
  Allocates an object from a memory pool.
- void * chPoolAlloc (memory_pool_t *mp)
  Allocates an object from a memory pool.
- void chPoolFreeI (memory_pool_t *mp, void *objp)
  Releases an object into a memory pool.
- void chPoolFree (memory_pool_t *mp, void *objp)
  Releases an object into a memory pool.
- void chGuardedPoolObjectInitAligned (guarded_memory_pool_t *gmp, size_t size, unsigned align)
  Initializes an empty guarded memory pool.
- void chGuardedPoolLoadArray (guarded_memory_pool_t *gmp, void *p, size_t n)
loads a guarded memory pool with an array of static objects.

- void chGuardedPoolAllocTimeoutS (guarded_memory_pool_t gmp, sysinterval_t timeout)
  Allocates an object from a guarded memory pool.

- void chGuardedPoolAllocTimeout (guarded_memory_pool_t gmp, sysinterval_t timeout)
  Allocates an object from a guarded memory pool.

- void chGuardedPoolFree (guarded_memory_pool_t gmp, void *objp)
  Releases an object into a guarded memory pool.

- static void chPoolObjectInit (memory_pool_t mp, size_t size, memgetfunc_t provider)
  Initializes an empty memory pool.

- static void chPoolAdd (memory_pool_t mp, void *objp)
  Adds an object to a memory pool.

- static void chPoolAddI (memory_pool_t mp, void *objp)
  Adds an object to a memory pool.

- static void chGuardedPoolObjectInit (guarded_memory_pool_t gmp, size_t size)
  Initializes an empty guarded memory pool.

- static cnt_t chGuardedPoolGetCounterI (guarded_memory_pool_t gmp)
  Gets the count of objects in a guarded memory pool.

- static void chGuardedPoolAllocI (guarded_memory_pool_t gmp)
  Allocates an object from a guarded memory pool.

- static void chGuardedPoolFreeI (guarded_memory_pool_t gmp, void *objp)
  Releases an object into a guarded memory pool.

- static void chGuardedPoolFreeS (guarded_memory_pool_t gmp, void *objp)
  Releases an object into a guarded memory pool.

- static void chGuardedPoolAdd (guarded_memory_pool_t gmp, void *objp)
  Adds an object to a guarded memory pool.

- static void chGuardedPoolAddI (guarded_memory_pool_t gmp, void *objp)
  Adds an object to a guarded memory pool.

- static void chGuardedPoolAddS (guarded_memory_pool_t gmp, void *objp)
  Adds an object to a guarded memory pool.

8.33.1 Detailed Description

Memory Pools macros and structures.

8.34 chmsg.c File Reference

Messages code.
#include "ch.h"

Functions

- msg_t chMsgSend (thread_t *tp, msg_t msg)
  Sends a message to the specified thread.

- thread_t *chMsgWaitS (void)
  Suspends the thread and waits for an incoming message.

- thread_t *chMsgWaitTimeoutS (sysinterval_t timeout)
  Suspends the thread and waits for an incoming message or a timeout to occur.

- thread_t *chMsgPollIS (void)
  Poll to check for an incoming message.

- void chMsgRelease (thread_t *tp, msg_t msg)
  Releases a sender thread specifying a response message.
8.34.1 Detailed Description

Messages code.

8.35 chmsg.h File Reference

Messages macros and structures.

Functions

- msg_t chMsgSend (thread_t *tp, msg_t msg)
  Sends a message to the specified thread.
- thread_t * chMsgWaitS (void)
  Suspends the thread and waits for an incoming message.
- thread_t * chMsgWaitTimeoutS (sysinterval_t timeout)
  Suspends the thread and waits for an incoming message or a timeout to occur.
- thread_t * chMsgPollS (void)
  Poll to check for an incoming message.
- void chMsgRelease (thread_t *tp, msg_t msg)
  Releases a sender thread specifying a response message.
- static thread_t * chMsgWait (void)
  Suspends the thread and waits for an incoming message.
- static thread_t * chMsgWaitTimeout (sysinterval_t timeout)
  Suspends the thread and waits for an incoming message or a timeout to occur.
- static thread_t * chMsgPoll (void)
  Poll to check for an incoming message.
- static bool chMsgIsPendingI (thread_t *tp)
  Evaluates to true if the thread has pending messages.
- static msg_t chMsgGet (thread_t *tp)
  Returns the message carried by the specified thread.
- static void chMsgReleaseS (thread_t *tp, msg_t msg)
  Releases the thread waiting on top of the messages queue.

8.35.1 Detailed Description

Messages macros and structures.

8.36 chmtx.c File Reference

Mutexes code.

#include "ch.h"
Functions

- void chMtxObjectInit (mutex_t *mp)
  Initializes a mutex 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.

8.36.1 Detailed Description

Mutexes code.

8.37 chmtx.h File Reference

Mutexes macros and structures.

Data Structures

- struct ch_mutex
  Mutex structure.

Macros

- #define __MUTEX_DATA(name) {__CH_QUEUE_DATA(name.queue), NULL, NULL, 0}
  Data part of a static mutex initializer.
- #define MUTEXDECL(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.

8.37.1 Detailed Description

Mutexes macros and structures.

8.38 chobjcaches.c File Reference

Objects Caches code.

#include "ch.h"

Functions

- static oc_object_t * hash_get_s (objects_cache_t *ocp, uint32_t group, uint32_t key)
  Returns an object pointer from the cache, if present.
- static oc_object_t * lru_get_last_s (objects_cache_t *ocp)
  Gets the least recently used object buffer from the LRU list.
- void chCacheObjectInit (objects_cache_t *ocp, ucnt_t hashn, oc_hash_header_t *hashp, ucnt_t objn, size_t objsz, void *objvp, oc_readf_t readf, oc_writef_t writef)
  Initializes a objects_cache_t object.
- oc_object_t * chCacheGetObject (objects_cache_t *ocp, uint32_t group, uint32_t key)
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.

8.38.1 Detailed Description

Objects Caches code.

Objects caches.

Operation mode

An object cache allows to retrieve and release objects from a slow media, for example a disk or flash. The most recently used objects are kept in a series of RAM buffers making access faster. Objects are identified by a pair \(<\text{group, key}\>\) which could be mapped, for example, to a disk drive identifier and sector identifier.

Read and write operations are performed using externally-supplied functions, the cache is device-agnostic. The cache uses internally an hash table, the size of the table should be dimensioned to minimize the risk of hash collisions, a factor of two is usually acceptable, it depends on the specific application requirements.

Operations defined for caches:

- Get Object: Retrieves an object from cache, if not present then an empty buffer is returned.
- Read Object: Retrieves an object from cache, if not present a buffer is allocated and the object is read from the media.
- Release Object: Releases an object to the cache handling the media update, if required.

Precondition

In order to use the pipes APIs the CH_CFG_USE_OBJ_CACHES option must be enabled in chconf.h.

Note

Compatible with RT and NIL.

8.39 chobjcaches.h File Reference

Objects Caches macros and structures.
Data Structures

- **struct ch_oc_hash_header**
  
  Structure representing an hash table element.

- **struct ch_oc_lru_header**
  
  Structure representing an 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.
8.39.1 Detailed Description

Objects Caches macros and structures.

8.40 chobjects.h File Reference

Operating System Objects macros and structures.

Data Structures

- struct ch_virtual_timer
  Structure representing a Virtual Timer.
- struct ch_virtual_timers_list
  Type of virtual timers list header.
- struct ch_registry
  Type of a registry structure.
- struct ch_threads_queue
  Type of a threads queue.
- struct ch_thread
  Structure representing a thread.
- struct ch_ready_list
  Type of a ready list header.
- struct ch_os_instance_config
  Type of an system instance configuration.
- struct ch_os_instance
  System instance data structure.
- struct ch_system
  Type of system data structure.

Typedefs

- typedef struct ch_virtual_timer virtual_timer_t
  Type of a Virtual Timer.
- typedef void(* vtfunc_t)(virtual_timer_t *vtp, void *p)
  Type of a Virtual Timer callback function.
- typedef struct ch_virtual_timers_list virtual_timers_list_t
  Type of virtual timers list header.
- typedef struct ch_registry registry_t
  Type of a registry structure.
- typedef thread_t * thread_reference_t
  Type of a thread reference.
- typedef struct ch_threads_queue threads_queue_t
  Type of a threads queue.
- typedef struct ch_ready_list ready_list_t
  Type of a ready list header.
- typedef struct ch_os_instance_config os_instance_config_t
  Type of an system instance configuration.
- typedef struct ch_system ch_system_t
  Type of system data structure.
Enumerations

- enum system_state_t
  
  Global state of the operating system.

8.40.1 Detailed Description

Operating System Objects macros and structures.

8.41 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.
8.41.1 Detailed Description

Objects FIFO structures and macros.

This module implements a generic FIFO queue of objects by coupling a Guarded Memory Pool (for objects storage) and a MailBox.

On the sender side free objects are taken from the pool, filled and then sent to the receiver, on the receiver side objects are fetched, used and then returned to the pool. Operations defined for object FIFOs:

- **Take**: An object is taken from the pool of the free objects, can be blocking.
- **Return**: An object is returned to the pool of the free objects, it is guaranteed to be non-blocking.
- **Send**: An object is sent through the mailbox, it is guaranteed to be non-blocking
- **Receive**: An object is received from the mailbox, can be blocking.

8.42 chpipes.c File Reference

Pipes code.

```c
#include <string.h>
#include "ch.h"
```

Functions

- static size_t pipe_write (pipe_t *pp, const uint8_t *bp, size_t n)
  Non-blocking pipe write.
- static size_t pipe_read (pipe_t *pp, uint8_t *bp, size_t n)
  Non-blocking pipe read.
- void chPipeObjectInit (pipe_t *pp, uint8_t *buf, size_t n)
  Initializes a mailbox_t object.
- void chPipeReset (pipe_t *pp)
  Resets a pipe_t object.
- size_t chPipeWriteTimeout (pipe_t *pp, const uint8_t *bp, size_t n, sysinterval_t timeout)
  Pipe write with timeout.
- size_t chPipeReadTimeout (pipe_t *pp, uint8_t *bp, size_t n, sysinterval_t timeout)
  Pipe read with timeout.
8.42.1 Detailed Description

Pipes code.

Byte pipes.

Operation mode

A pipe is an asynchronous communication mechanism. Operations defined for mailboxes:

- **Write**: Writes a buffer of data in the pipe in FIFO order.
- **Read**: A buffer of data is read from the read and removed.
- **Reset**: The pipe is emptied and all the stored data is lost.

Precondition

In order to use the pipes APIs the `CH_CFG_USE_PIPES` option must be enabled in `chconf.h`.

Note

Compatible with RT and NIL.

8.43 chpipes.h File Reference

Pipes macros and structures.

Data Structures

- `struct pipe_t`
  
  Structure representing a pipe object.

Macros

- `#define __PIPE_DATA(name, buffer, size)`
  
  Data part of a static pipe initializer.
- `#define PIPE_DECL(name, buffer, size) pipe_t name = __PIPE_DATA(name, buffer, size)`
  
  Static pipe initializer.
Functions

- void chPipeObjectInit (pipe_t *pp, uint8_t *buf, size_t n)
  
  Initializes a mailbox_t object.
- void chPipeReset (pipe_t *pp)
  
  Resets a pipe_t object.
- size_t chPipeWriteTimeout (pipe_t *pp, const uint8_t *bp, size_t n, sysinterval_t timeout)
  
  Pipe write with timeout.
- size_t chPipeReadTimeout (pipe_t *pp, uint8_t *bp, size_t n, sysinterval_t timeout)
  
  Pipe read with timeout.
- static size_t chPipeGetSize (const pipe_t *pp)
  
  Returns the pipe buffer size as number of bytes.
- static size_t chPipeGetUsedCount (const pipe_t *pp)
  
  Returns the number of used byte slots into a pipe.
- static size_t chPipeGetFreeCount (const pipe_t *pp)
  
  Returns the number of free byte slots into a pipe.
- static void chPipeResume (pipe_t *pp)
  
  Terminates the reset state.

8.43.1 Detailed Description

Pipes macros and structures.

8.44 chport.h File Reference

Port wrapper header.

#include "chcore.h"

8.44.1 Detailed Description

Port wrapper header.

8.45 chregistry.c File Reference

Threads registry code.

#include <string.h>
#include "ch.h"
8.46 chregistry.h File Reference

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.

8.45.1 Detailed Description

Threads registry code.

8.46 chregistry.h File Reference

Threads registry macros and structures.

Data Structures

- `struct chdebug_t`
  
  ChibiOS/RT memory signature record.

Macros

- `#define REG_HEADER(oip) (&ch_system.reglist.queue)`
  
  Access to the registry list header.

- `#define REG_REMOVE(tp) (void) ch_queue_dequeue(&(tp)->rqueue)`
  
  Removes a thread from the registry list.

- `#define REG_INSERT(oip, tp) ch_queue_insert(REG_HEADER(oip), &(tp)->rqueue)`
  
  Adds a thread to the registry list.
Functions

- thread_t * chRegFirstThread (void)
  Returns the first thread in the system.
- thread_t * chRegNextThread (thread_t *tp)
  Returns the thread next to the specified one.
- thread_t * chRegFindThreadByName (const char *name)
  Retrieves a thread pointer by name.
- thread_t * chRegFindThreadByPointer (thread_t *tp)
  Confirms that a pointer is a valid thread pointer.
- thread_t * chRegFindThreadByWorkingArea (stkalign_t *wa)
  Confirms that a working area is being used by some active thread.
- static void __reg_object_init (registry_t *rp)
  Initializes a registry.
- static void chRegSetThreadName (const char *name)
  Sets the current thread name.
- static const char * chRegGetThreadNameX (thread_t *tp)
  Returns the name of the specified thread.
- static void chRegSetThreadNameX (thread_t *tp, const char *name)
  Changes the name of the specified thread.

8.46.1 Detailed Description

Threads registry macros and structures.

8.47 chrestrictions.h File Reference

Licensing restrictions header.

8.47.1 Detailed Description

Licensing restrictions header.

8.48 chrfcu.c File Reference

Runtime Faults Collection Unit code.

#include "ch.h"

Functions

- void chRFCUCollectFaultsI (rfcu_mask_t mask)
  Adds fault flags to the current mask.
- rfcu_mask_t chRFCUGetAndClearFaultsI (rfcu_mask_t mask)
  Returns the current faults mask clearing it.
8.49 chrfcu.h File Reference

Runtime Faults Collection Unit macros and structures.

Data Structures

- struct ch_rfcu
  
  
  Type of an RFCU structure.

Macros

- #define CH_RFCU_ALL_FAULTS ((rfcu_mask_t)-1)
  
  Mask of all faults.

Predefined Faults

- #define CH_RFCU_VT_INSUFFICIENT_DELTA 1U
- #define CH_RFCU_VT_SKIPPED_DEADLINE 2U

Typedefs

- typedef uint32_t rfcu_mask_t
  
  Type of a faults mask.
- typedef struct ch_rfcu rfcu_t
  
  Type of an RFCU structure.

Functions

- void chRFCUCollectFaultsI (rfcu_mask_t mask)
  
  Adds fault flags to the current mask.
- rfcu_mask_t chRFCUGetAndClearFaultsI (rfcu_mask_t mask)
  
  Returns the current faults mask clearing it.
- static void __rfcu_object_init (rfcu_t *rfcup)
  
  Runtime Faults Collection Unit initialization.

8.49.1 Detailed Description

Runtime Faults Collection Unit macros and structures.
8.50  chscht.c File Reference

Scheduler code.

#include "ch.h"

Functions

- static thread_t * __sch_ready_behind (thread_t *tp)
  Inserts a thread in the Ready List placing it behind its peers.
- static thread_t * __sch_ready_ahead (thread_t *tp)
  Inserts a thread in the Ready List placing it ahead its peers.
- static void __sch_reschedule_behind (void)
  Switches to the first thread on the runnable queue.
- static void __sch_reschedule_ahead (void)
  Switches to the first thread on the runnable queue.
- void ch_sch_prio_insert (ch_queue_t *qp, ch_queue_t *tp)
  Inserts a thread into a priority ordered queue.
- thread_t * chSchReadyI (thread_t *tp)
  Inserts a thread in the Ready List placing it behind its peers.
- void chSchGoSleepS (tstate_t newstate)
  Puts the current thread to sleep into the specified state.
- msg_t chSchGoSleepTimeoutS (tstate_t newstate, sysinterval_t timeout)
  Puts the current thread to sleep into the specified state with timeout specification.
- void chSchWakeupS (thread_t *ntp, msg_t msg)
  Wakes up a thread.
- void chSchRescheduleS (void)
  Performs a reschedule if a higher priority thread is runnable.
- bool chSchIsPreemptionRequired (void)
  Evaluates if preemption is required.
- void chSchDoPreemption (void)
  Switches to the first thread on the runnable queue.
- void chSchPreemption (void)
  All-in-one preemption code.
- void chSchDoYieldS (void)
  Yields the time slot.
- thread_t * chSchSelectFirst (void)
  Makes runnable the fist thread in the ready list, does not reschedule internally.

8.50.1  Detailed Description

Scheduler code.

8.51  chschd.h File Reference

Scheduler macros and structures.
Macros

- `#define firstprio(rlp) ((rlp)->next->prio)`
  Returns the priority of the first thread on the given ready list.
- `#define __sch_get_currthread() __instance_get_currthread(currcore)`
  Current thread pointer get macro.

Wakeup status codes

- `#define MSG_OK (msg_t)0`
  Normal wakeup message.
- `#define MSG_TIMEOUT (msg_t)-1`
  Wakeup caused by a timeout condition.
- `#define MSG_RESET (msg_t)-2`
  Wakeup caused by a reset condition.

Priority constants

- `#define NOPRIO (tprio_t)0`
  Ready list header priority.
- `#define IDLEPRIO (tprio_t)1`
  Idle priority.
- `#define LOWPRIO (tprio_t)2`
  Lowest priority.
- `#define NORMALPRIO (tprio_t)128`
  Normal priority.
- `#define HIGHPRIO (tprio_t)255`
  Highest priority.

Thread states

- `#define CH_STATE_READY (tstate_t)0`
  Waiting on the ready list.
- `#define CH_STATE_CURRENT (tstate_t)1`
  Currently running.
- `#define CH_STATE_WTSTART (tstate_t)2`
  Just created.
- `#define CH_STATE_SUSPENDED (tstate_t)3`
  Suspended state.
- `#define CH_STATE_QUEUED (tstate_t)4`
  On a queue.
- `#define CH_STATE_WTSEM (tstate_t)5`
  On a semaphore.
- `#define CH_STATE_WTMTX (tstate_t)6`
On a mutex.

- `#define CH_STATE_WTCOND (tstate_t)`
- `#define CH_STATE_SLEEPING (tstate_t)`
- `#define CH_STATE_WTEXIT (tstate_t)`
- `#define CH_STATE_WTOREVT (tstate_t)`
- `#define CH_STATE_WTANDEVT (tstate_t)`
- `#define CH_STATE_SNDMSGQ (tstate_t)`
- `#define CH_STATE_SNDMSG (tstate_t)`
- `#define CH_STATE_WTMSG (tstate_t)`
- `#define CH_STATE_FINAL (tstate_t)`

Thread states as array of strings.

Thread flags and attributes

- `#define CH_FLAG_MODE_MASK (tmode_t)`
- `#define CH_FLAG_MODE_STATIC (tmode_t)`
- `#define CH_FLAG_MODE_HEAP (tmode_t)`
- `#define CH_FLAG_MODE_MPOOL (tmode_t)`
- `#define CH_FLAG_TERMINATE (tmode_t)`

Thread memory mode mask.
Static thread.
Thread allocated from a Memory Heap.
Thread allocated from a Memory Pool.
Termination requested flag.

Functions

- `thread_t * chSchReadyI (thread_t *tp)`
  Inserts a thread in the Ready List placing it behind its peers.
- `void chSchGoSleepS (tstate_t newstate)`
  Puts the current thread to sleep into the specified state.
- `msg_t chSchGoSleepTimeoutS (tstate_t newstate, sysinterval_t timeout)`
  Puts the current thread to sleep into the specified state with timeout specification.
- `void chSchWakeupS (thread_t *ntp, msg_t msg)`
Wakes up a thread.

- void chSchRescheduleS (void)
  Performs a reschedule if a higher priority thread is runnable.

- bool chSchIsPreemptionRequired (void)
  Evaluates if preemption is required.

- void chSchDoPreemption (void)
  Switches to the first thread on the runnable queue.

- void chSchPreemption (void)
  All-in-one preemption code.

- void chSchDoYieldS (void)
  Yields the time slot.

- thread_t * chSchSelectFirst (void)
  Makes runnable the first thread in the ready list, does not reschedule internally.

- void ch_sch_prio_insert (ch_queue_t * qp, ch_queue_t * tp)
  Inserts a thread into a priority ordered queue.

8.51.1 Detailed Description

Scheduler macros and structures.

8.52 chsem.c File Reference

Semaphores code.

#include "ch.h"

Functions

- void chSemObjectInit (semaphore_t * sp, cnt_t n)
  Initializes a semaphore with the specified counter value.

- void chSemResetWithMessage (semaphore_t * sp, cnt_t n, msg_t msg)
  Performs a reset operation on the semaphore.

- void chSemResetWithMessageI (semaphore_t * sp, cnt_t n, msg_t msg)
  Performs a reset operation on the semaphore.

- msg_t chSemWait (semaphore_t * sp)
  Performs a wait operation on a semaphore.

- msg_t chSemWaitS (semaphore_t * sp)
  Performs a wait operation on a semaphore.

- msg_t chSemWaitTimeout (semaphore_t * sp, sysinterval_t timeout)
  Performs a wait operation on a semaphore with timeout specification.

- msg_t chSemWaitTimeoutS (semaphore_t * sp, sysinterval_t timeout)
  Performs a wait operation on a semaphore with timeout specification.

- void chSemSignal (semaphore_t * sp)
  Performs a signal operation on a semaphore.

- void 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.

- void chSemAddCounter (semaphore_t * sp, cnt_t n)
  Adds the specified value to the semaphore counter.

- void chSemAddCounterI (semaphore_t * sp, cnt_t n)
  Adds the specified value to the semaphore counter.

- void chSemAddCounter (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.
8.52.1 Detailed Description

Semaphores code.

8.53 chsem.h File Reference

Semaphores macros and structures.

Data Structures

- struct ch_semaphore
  
  Semaphore structure.

Macros

- #define __SEMAPHORE_DATA(name, n) {__CH_QUEUE_DATA(name.queue), n}
  
  Data part of a static semaphore initializer.
- #define SEMAPHORE_DECL(name, n) semaphore_t name = __SEMAPHORE_DATA(name, n)
  
  Static semaphore initializer.

Typedefs

- typedef struct ch_semaphore semaphore_t
  
  Semaphore structure.

Functions

- void chSemObjectInit (semaphore_t *sp, cnt_t n)
  
  Initializes a semaphore with the specified counter value.
- void chSemResetWithMessage (semaphore_t *sp, cnt_t n, msg_t msg)
  
  Performs a reset operation on the semaphore.
- void 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)
8.54 chstats.c File Reference

Statistics module code.

#include "ch.h"

Functions

• void __stats_increase_irq (void)
  Increases the IRQ counter.
• void __stats_ctxswc (thread_t *ntp, thread_t *otp)
  Updates context switch related statistics.
• void __stats_start_measure_crit_thd (void)
  Starts the measurement of a thread critical zone.
• void __stats_stop_measure_crit_thd (void)
  Stops the measurement of a thread critical zone.
• void __stats_start_measure_crit_isr (void)
  Starts the measurement of an ISR critical zone.
• void __stats_stop_measure_crit_isr (void)
  Stops the measurement of an ISR critical zone.

8.54.1 Detailed Description

Statistics module code.
8.55  chstats.h File Reference

Statistics module macros and structures.

Data Structures

- struct kernel_stats_t
  
  Type of a kernel statistics structure.

Functions

- void __stats_increase_irq (void)
  
  Increases the IRQ counter.

- void __stats_ctxswc (thread_t *ntp, thread_t *otp)
  
  Updates context switch related statistics.

- void __stats_start_measure_crit_thd (void)
  
  Starts the measurement of a thread critical zone.

- void __stats_stop_measure_crit_thd (void)
  
  Stops the measurement of a thread critical zone.

- void __stats_start_measure_crit_isr (void)
  
  Starts the measurement of an ISR critical zone.

- void __stats_stop_measure_crit_isr (void)
  
  Stops the measurement of an ISR critical zone.

- static void __stats_object_init (kernel_stats_t *ksp)
  
  Statistics initialization.

8.55.1  Detailed Description

Statistics module macros and structures.

8.56  chsys.c File Reference

System related code.

#include "ch.h"
Functions

- static CH_SYS_CORE0_MEMORY THD_WORKING_AREA (ch_c0_idle_thread_wa, PORT_IDLE_THRE ← AD_STACK_SIZE)
  
  Working area for core 0 idle thread.
- static CH_SYS_CORE1_MEMORY THD_WORKING_AREA (ch_c1_idle_thread_wa, PORT_IDLE_THRE ← AD_STACK_SIZE)
  
  Working area for core 1 idle thread.
- void chSysWaitSystemState (system_state_t state)
  
  Waits for the system state to be equal to the specified one.
- void chSysInit (void)
  
  System initialization.
- void chSysHalt (const char ∗reason)
  
  Halts the system.
- thread_t ∗ chSysGetIdleThreadX (void)
  
  Returns a pointer to the idle thread.
- 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.

Variables

- ch_system_t ch_system
  
  System root object.
- CH_SYS_CORE0_MEMORY os_instance_t ch0
  
  Core 0 OS instance.
- const os_instance_config_t ch_core0_cfg
  
  Core 0 OS instance configuration.
- CH_SYS_CORE1_MEMORY os_instance_t ch1
  
  Core 1 OS instance.
- const os_instance_config_t ch_core1_cfg
  
  Core 1 OS instance configuration.

8.56.1 Detailed Description

System related code.

8.57 chsys.h File Reference

System related macros and structures.
Macros

- #define CH_SYS_CORE0_MEMORY PORT_CORE0_BSS_SECTION
  Core zero memory affinity macro.
- #define CH_SYS_CORE1_MEMORY PORT_CORE1_BSS_SECTION
  Core one memory affinity macro.
- #define currcore ch_system.instances[port_get_core_id()]
  Access to current core's instance structure.
- #define chSysGetRealtimeCounterX() (rtcnt_t)port_rt_get_counter_value()
  Returns the current value of the system real time counter.
- #define chSysSwitch(ntp, otp)
  Performs a context switch.

Masks of executable integrity checks.

- #define CH_INTEGRITY_RLIST 1U
- #define CH_INTEGRITY_VTLIST 2U
- #define CH_INTEGRITY_REGISTRY 4U
- #define CH_INTEGRITY_PORT 8U

ISRs abstraction macros

- #define CH_IRQ_IS_VALID_PRIORITY(prio) PORT_IRQ_IS_VALID_PRIORITY(prio)
  Priority level validation macro.
- #define CH_IRQ_IS_VALID_KERNEL_PRIORITY(prio) PORT_IRQ_IS_VALID_KERNEL_PRIORITY(prio)
  Priority level validation macro.
- #define CH_IRQ_PROLOGUE()
  IRQ handler enter code.
- #define CH_IRQ_EPILOGUE()
  IRQ handler exit code.
- #define CH_IRQ_HANDLER(id) PORT_IRQ_HANDLER(id)
  Standard normal IRQ handler declaration.

Fast ISRs abstraction macros

- #define CH_FAST_IRQ_HANDLER(id) PORT_FAST_IRQ_HANDLER(id)
  Standard fast IRQ handler declaration.

Time conversion utilities for the realtime counter

- #define S2RTC(freq, sec) (((freq) * (sec))
  Seconds to realtime counter.
- #define MS2RTC(freq, msec) (rtcnt_t)(((freq) + 999UL) / 1000UL) * (msec)
  Milliseconds to realtime counter.
- #define US2RTC(freq, usec) (rtcnt_t)(((freq) + 999999UL) / 1000000UL) * (usec)
  Microseconds to realtime counter.
- #define RTC2S(freq, n) ((((n) - 1UL) / (freq)) + 1UL)
  Realtime counter cycles to seconds.
- #define RTC2MS(freq, n) ((((n) - 1UL) / (freq) / 1000UL)) + 1UL
  Realtime counter cycles to milliseconds.
- #define RTC2US(freq, n) ((((n) - 1UL) / (freq) / 1000000UL)) + 1UL
  Realtime counter cycles to microseconds.
Functions

- void chSysWaitSystemState (system_state_t state)
  Waits for the system state to be equal to the specified one.
- void chSysInit (void)
  System initialization.
- thread_t * chSysGetIdleThreadX (void)
  Returns a pointer to the idle thread.
- 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 void chSysNotifyInstance (os_instance_t *oip)
  Notifies an OS instance to check for reschedule.

8.57.1 Detailed Description

System related macros and structures.

8.58 chthreads.c File Reference

Threads code.

#include "ch.h"
Functions

- thread_t * __thd_object_init (os_instance_t *oip, thread_t *tp, const char *name, tprio_t prio)
  
  Initializes a thread structure.

- void __thd_stackfill (uint8_t *startp, uint8_t *endp)
  
  Stack 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 * chThdCreate (const thread_descriptor_t *tdp)
  
  Creates a new thread into a static memory area.

- thread_t * chThdCreateStatic (void *wsp, size_t size, tprio_t prio, tfunc_t pf, void *arg)
  
  Creates a new thread into a static memory area.

- thread_t * chThdStart (thread_t *tp)
  
  Resumes a thread created with chThdCreateI().

- thread_t * chThdAddRef (thread_t *tp)
  
  Adds a reference to a thread object.

- void chThdRelease (thread_t *tp)
  
  Releases a reference to a thread object.

- void chThdExit (msg_t msg)
  
  Terminates the current thread.

- void chThdExitS (msg_t msg)
  
  Terminates the current thread.

- msg_t chThdWait (thread_t *tp)
  
  Blocks the execution of the invoking thread until the specified thread terminates then the exit code is returned.

- tprio_t chThdSetPriority (tprio_t newprio)
  
  Changes the running thread priority level then reschedules if necessary.

- void chThdTerminate (thread_t *tp)
  
  Requests a thread termination.

- void chThdSleep (sysinterval_t time)
  
  Suspends the invoking thread for the specified time.

- void chThdSleepUntil (systime_t time)
  
  Suspends the invoking thread until the system time arrives to the specified value.

- systime_t chThdSleepUntilWindowed (systime_t prev, systime_t next)
  
  Suspends the invoking thread until the system time arrives to the specified value.

- void chThdYield (void)
  
  Yields the time slot.

- msg_t chThdSuspendS (thread_reference_t *trp)
  
  Sends the current thread sleeping and sets a reference variable.

- msg_t chThdSuspendTimeoutS (thread_reference_t *trp, sysinterval_t timeout)
  
  Sends the current thread sleeping and sets a reference variable.

- void chThdResumeI (thread_reference_t *trp, msg_t msg)
  
  Wakes up a thread waiting on a thread reference object.

- void chThdResumeS (thread_reference_t *trp, msg_t msg)
  
  Wakes up a thread waiting on a thread reference object.

- void chThdResume (thread_reference_t *trp, msg_t msg)
  
  Wakes up a thread waiting on a thread reference object.

- msg_t chThdEnqueueTimeoutS (threads_queue_t *tqp, sysinterval_t timeout)
Enqueues the caller thread on a threads queue object.

- void chThdDequeueNextI (threads_queue_t *tp, msg_t msg)
  Dequeues and wakes up one thread from the threads queue object, if any.
- void chThdDequeueAllI (threads_queue_t *tp, msg_t msg)
  Dequeues and wakes up all threads from the threads queue object.

8.58.1 Detailed Description

Threads code.

8.59 chthreads.h File Reference

Threads module macros and structures.

Data Structures

- struct thread_descriptor_t
  Type of a thread descriptor.

Macros

Threads queues

- #define __THREADS_QUEUE_DATA(name) {__CH_QUEUE_DATA(name)}
  Data part of a static threads queue object initializer.
- #define THREADS_QUEUE_DECL(name) threads_queue_t name = __THREADS_QUEUE_DATA(name)
  Static threads queue object initializer.

Working Areas

- #define THD_WORKING_AREA_SIZE(n) MEM_ALIGN_NEXT(sizeof(thread_t) + PORT_WA_SIZE(n), PORT_STACK_ALIGN)
  Calculates the total Working Area size.
- #define THD_WORKING_AREA(s, n) PORT_WORKING_AREA(s, n)
  Static working area allocation.
- #define THD_WORKING_AREA_BASE(s) ((stkalign_t *)(s))
  Base of a working area casted to the correct type.
- #define THD_WORKING_AREA_END(s)
  End of a working area casted to the correct type.

Threads abstraction macros

- #define THD_FUNCTION(tname, arg) PORT_THD_FUNCTION(tname, arg)
  Thread declaration macro.

Threads initializers

- #define THD_DESCRIPTOR(name, wbase, wend, prio, funcp, arg)
  Thread descriptor initializer with no affinity.
```c
• #define THD_DESCRIPTOR_AFFINITY(name, wbase, wend, prio, funcp, arg, oip)
    Thread descriptor initializer with no affinity.

Macro Functions

• #define chThdSleepSeconds(sec) chThdSleep(TIME_S2I(sec))
    Delays the invoking thread for the specified number of seconds.
• #define chThdSleepMilliseconds(msec) chThdSleep(TIME_MS2I(msec))
    Delays the invoking thread for the specified number of milliseconds.
• #define chThdSleepMicroseconds(usec) chThdSleep(TIME_US2I(usec))
    Delays the invoking thread for the specified number of microseconds.

Typedefs

• typedef void(*tfunc_t)(void*p)
    Thread function.

Functions

• thread_t *__thd_object_init (os_instance_t* oip, thread_t* tp, const char* name, tprio_t prio)
    Initializes a thread structure.
• void __thd_stackfill (uint8_t* startp, uint8_t* endp)
    Stack fill utility.
• thread_t* chThdCreateSuspendedI (const thread_descriptor_t* tdp)
    Creates a new thread into a static memory area.
• thread_t* chThdCreateSuspended (const thread_descriptor_t* tdp)
    Creates a new thread into a static memory area.
• thread_t* chThdCreateI (const thread_descriptor_t* tdp)
    Creates a new thread into a static memory area.
• thread_t* chThdCreate (const thread_descriptor_t* tdp)
    Creates a new thread into a static memory area.
• thread_t* chThdCreateStatic (void*wsp, size_t size, tprio_t prio, tfunc_t pf, void* arg)
    Creates a new thread into a static memory area.
• thread_t* chThdStart (thread_t* tp)
    Resumes a thread created with chThdCreateI().
• thread_t* chThdAddRef (thread_t* tp)
    Adds a reference to a thread object.
• void chThdRelease (thread_t* tp)
    Releases a reference to a thread object.
• void chThdExit (msg_t msg)
    Terminates the current thread.
• void chThdExitS (msg_t msg)
    Terminates the current thread.
• msg_t chThdWait (thread_t* tp)
    Blocks the execution of the invoking thread until the specified thread terminates then the exit code is returned.
• tprio_t chThdSetPriority (tprio_t newprio)
    Changes the running thread priority level then reschedules if necessary.
• void chThdTerminate (thread_t* tp)
    Requests a thread termination.
• msg_t chThdSuspendS (thread_reference_t* trp)
    Suspends the specified thread.
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.

• void chThdSleep (sysinterval_t time)
  Suspends the invoking thread for the specified time.

• void chThdSleepUntil (systime_t time)
  Suspends the invoking thread until the system time arrives to the specified value.

• systime_t chThdSleepUntilWindowed (systime_t prev, systime_t next)
  Suspends the invoking thread until the system time arrives to the specified value.

• void chThdYield (void)
  Yields the time slot.

• static thread_t * chThdGetSelfX (void)
  Returns a pointer to the current thread_t.

• static tprio_t chThdGetPriorityX (void)
  Returns the current thread priority.

• static systime_t chThdGetTicksX (thread_t *tp)
  Returns the number of ticks consumed by the specified thread.

• static stkalign_t * chThdGetWorkingAreaX (thread_t *tp)
  Returns the working area base of the specified thread.

• static bool chThdTerminatedX (thread_t *tp)
  Verifies if the specified thread is in the CH_STATE_FINAL state.

• static bool chThdShouldTerminateX (void)
  Verifies if the current thread has a termination request pending.

• static thread_t * 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 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.

8.59.1 Detailed Description

Threads module macros and structures.
8.60  chtime.h File Reference

Time and intervals macros and structures.

Macros

Special time constants

- \#define TIME_IMMEDIATE ((sysinterval_t)0)
  Zero interval specification for some functions with a timeout specification.
- \#define TIME_INFINITE ((sysinterval_t)-1)
  Infinite interval specification for all functions with a timeout specification.
- \#define TIME_MAX_INTERVAL ((sysinterval_t)-2)
  Maximum interval constant usable as timeout.
- \#define TIME_MAX_SYSTIME ((systime_t)-1)
  Maximum system of system time before it wraps.

Fast time conversion utilities

- \#define TIME_S2I(secs) ((sysinterval_t)((time_conv_t)(secs) * (time_conv_t)CH_CFG_ST_FREQUENCY))
  Seconds to time interval.
- \#define TIME_MS2I(msecs)
  Milliseconds to time interval.
- \#define TIME_US2I(usecs)
  Microseconds to time interval.
- \#define TIME_I2S(interval)
  Time interval to seconds.
- \#define TIME_I2MS(interval)
  Time interval to milliseconds.
- \#define TIME_I2US(interval)
  Time interval to microseconds.

Typedefs

- typedef uint64_t systime_t
  Type of system time.
- typedef uint64_t sysinterval_t
  Type of time interval.
- typedef uint64_t systimestamp_t
  Type of a time stamp.
- typedef uint32_t time_secs_t
  Type of seconds.
- typedef uint32_t time_msecs_t
  Type of milliseconds.
- typedef uint32_t time_usecs_t
  Type of microseconds.
- typedef uint64_t time_conv_t
  Type of time conversion variable.
Functions

Secure time conversion utilities

- static sysinterval_t chTimeS2I (time_secs_t secs)
  Seconds to time interval.
- static sysinterval_t chTimeMS2I (time_msecs_t msec)
  Milliseconds to time interval.
- static sysinterval_t chTimeUS2I (time_usecs_t usec)
  Microseconds to time interval.
- static time_secs_t 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 chTimeI2S (sysinterval_t interval)
  Time interval to seconds.
- static time_usecs_t chTimeAddX (time_usecs_t usec)
  Adds an interval to a system time returning a system time.

8.60.1 Detailed Description

Time and intervals macros and structures.

8.61 chtm.c File Reference

Time Measurement module code.

#include "ch.h"

Functions

- void chTMObjectInit (time_measurement_t *tmp)
  Initializes a TimeMeasurement object.
- NOINLINE void chTMStartMeasurementX (time_measurement_t *tmp)
  Starts a measurement.
- NOINLINE void chTMStopMeasurementX (time_measurement_t *tmp)
  Stops a measurement.
- NOINLINE void chTMChainMeasurementToX (time_measurement_t *tmp1, time_measurement_t *tmp2)
  Stops a measurement and chains to the next one using the same time stamp.
8.61.1 Detailed Description

Time Measurement module code.

8.62 chtm.h File Reference

Time Measurement module macros and structures.

Data Structures

- `struct tm_calibration_t`
  
  Type of a time measurement calibration data.

- `struct time_measurement_t`
  
  Type of a Time Measurement object.

Macros

- `#define TM_CALIBRATION_LOOP 4U`
  
  Number of iterations in the calibration loop.

Functions

- `void chTMObjectInit (time_measurement_t *tmp)`
  
  Initializes a TimeMeasurement object.

- `NOINLINE void chTMStartMeasurementX (time_measurement_t *tmp)`
  
  Starts a measurement.

- `NOINLINE void chTMStopMeasurementX (time_measurement_t *tmp)`
  
  Stops a measurement.

- `NOINLINE void chTMChainMeasurementToX (time_measurement_t *tmp1, time_measurement_t *tmp2)`
  
  Stops a measurement and chains to the next one using the same time stamp.

- `static void __tm_calibration_object_init (tm_calibration_t *tcp)`
  
  Time measurement initialization.

8.62.1 Detailed Description

Time Measurement module macros and structures.

8.63 chtrace.c File Reference

Tracer code.

#include "ch.h"
8.64 chtrace.h File Reference

Tracer macros and structures.

Data Structures

- struct trace_event_t
  Trace buffer record.
- struct trace_buffer_t
  Trace buffer header.
Macros

Trace record types

- #define CH_TRACE_TYPE_UNUSED 0U
- #define CH_TRACE_TYPE_READY 1U
- #define CH_TRACE_TYPE_SWITCH 2U
- #define CH_TRACE_TYPE_ISR_ENTER 3U
- #define CH_TRACE_TYPE_ISR_LEAVE 4U
- #define CH_TRACE_TYPE_HALT 5U
- #define CH_TRACE_TYPE_USER 6U

Events to trace

- #define CH_DBG_TRACE_MASK_DISABLED 255U
- #define CH_DBG_TRACE_MASK_NONE 0U
- #define CH_DBG_TRACE_MASK_READY 1U
- #define CH_DBG_TRACE_MASK_SWITCH 2U
- #define CH_DBG_TRACE_MASK_ISR 4U
- #define CH_DBG_TRACE_MASK_HALT 8U
- #define CH_DBG_TRACE_MASK_USER 16U
- #define CH_DBG_TRACE_MASK_SLOW
- #define CH_DBG_TRACE_MASK_ALL

Debug related settings

- #define CH_DBG_TRACE_MASK CH_DBG_TRACE_MASK_DISABLED
- #define CH_DBG_TRACE_BUFFER_SIZE 128

Functions

- void __trace_object_init (trace_buffer_t *tbp)
  Circular trace buffer initialization.
- void __trace_ready (thread_t *tp, msg_t msg)
  Inserts in the circular debug trace buffer a ready record.
- void __trace_switch (thread_t *ntp, thread_t *otp)
  Inserts in the circular debug trace buffer a context switch record.
- void __trace_isr_enter (const char *isr)
  Inserts in the circular debug trace buffer an ISR-enter record.
- void __trace_isr_leave (const char *isr)
  Inserts in the circular debug trace buffer an ISR-leave record.
- void __trace_halt (const char *reason)
  Inserts in the circular debug trace buffer an halt record.
- void chTraceWriteI (void *up1, void *up2)
  Adds an user trace record to the trace buffer.
- void chTraceWrite (void *up1, void *up2)
  Adds an user trace record to the trace buffer.
- void chTraceSuspendI (uint16_t mask)
  Suspends one or more trace events.
- void chTraceSuspend (uint16_t mask)
  Suspends one or more trace events.
- void chTraceResume (uint16_t mask)
  Resumes one or more trace events.
8.65.1 Detailed Description

Tracer macros and structures.

8.65 chversion.h File Reference

Version Module macros and structures.

Macros

- `#define __CHIBIOS__`  
  ChibiOS product identification macro.
- `#define CH_VERSION_STABLE 1`  
  Stable release flag.
- `#define CH_VERSION_DATE (((CH_VERSION_YEAR + 2000) * 100) + CH_VERSION_MONTH)`  
  Current version date in numeric form (yyyymm).

ChibiOS version identification

- `#define CH_VERSION "2021.11.0"`  
  ChibiOS version string.
- `#define CH_VERSION_YEAR 21`  
  ChibiOS version release year.
- `#define CH_VERSION_MONTH 11`  
  ChibiOS version release month.
- `#define CH_VERSION_PATCH 0`  
  ChibiOS version patch number.
- `#define CH_VERSION_NICKNAME "Vettica"`  
  ChibiOS version nickname.

8.65.1 Detailed Description

Version Module macros and structures.

8.66 chvt.c File Reference

Time and Virtual Timers module code.

#include "ch.h"
Functions

- 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 virtual timer in a virtual timers list.
- void chVTDoSetI (virtual_timer_t *vtp, sysinterval_t delay, vtfunc_t vtfunc, void *par)
  
  Enables a one-shot virtual timer.
- void chVTDoSetContinuousI (virtual_timer_t *vtp, sysinterval_t delay, vtfunc_t vtfunc, void *par)
  
  Enables a continuous 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.
- systimestamp_t chVTGetTimeStampI (void)
  
  Generates a monotonic time stamp.
- void chVTResetTimeStampI (void)
  
  Resets and re-synchronizes the time stamps monotonic counter.

8.66.1 Detailed Description

Time and Virtual Timers module code.

8.67 chvt.h File Reference

Time and Virtual Timers module macros and structures.

Functions

- void chVTDoSetI (virtual_timer_t *vtp, sysinterval_t delay, vtfunc_t vtfunc, void *par)
  
  Enables a one-shot virtual timer.
- void chVTDoSetContinuousI (virtual_timer_t *vtp, sysinterval_t delay, vtfunc_t vtfunc, void *par)
  
  Enables a continuous 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.
- systimestamp_t chVTGetTimeStampI (void)
  
  Generates a monotonic time stamp.
- void chVTResetTimeStampI (void)
  
  Resets and re-synchronizes the time stamps monotonic counter.
8.67.1 Detailed Description

Time and Virtual Timers module macros and structures.
Index

__BSEMAPHORE_DATA
  Binary Semaphores, 351
__CHIBIOS_OSLIB__
  Version Numbers and Identification, 347
__CHIBIOS_RT__
  Version Numbers and Identification, 23
__CHIBIOS__
  Release Information, 16
__CH_OFFSETOF
  OS Types and Structures, 55
__CH_QUEUE_DATA
  Lists and Queues, 70
__CH_STRINGIFY
  OS Types and Structures, 53
__CH_USED
  OS Types and Structures, 55
__CONDVAR_DATA
  Condition Variables, 259
__EVENTSOURCE_DATA
  Event Flags, 272
__GUARDEDMEMORYPOOL_DATA
  Memory Pools, 436
__MAILBOX_DATA
  Mailboxes, 363
__MEMORYPOOL_DATA
  Memory Pools, 435
__MUTEX_DATA
  Mutexes, 244
__PIPE_DATA
  Pipes, 380
__SEMAPHORE_DATA
  Counting Semaphores, 226
__THREADS_QUEUE_DATA
  Threads, 180
__ch_delegate_fn0
  Delegate Threads, 389
__ch_delegate_fn1
  Delegate Threads, 390
__ch_delegate_fn2
  Delegate Threads, 390
__ch_delegate_fn3
  Delegate Threads, 390
__ch_delegate_fn4
  Delegate Threads, 392
__core_init
  Core Memory Manager, 419
__dbg_check_disable
  Checks and Assertions, 325
__dbg_check_enable
  Checks and Assertions, 325
__dbg_check_enter_isr
  Checks and Assertions, 329
__dbg_check_leave_isr
  Checks and Assertions, 329
__dbg_check_lock
  Checks and Assertions, 327
__dbg_check_lock_from_isr
  Checks and Assertions, 328
__dbg_check_suspend
  Checks and Assertions, 326
__dbg_check_unlock
  Checks and Assertions, 327
__dbg_check_unlock_from_isr
  Checks and Assertions, 328
__dbg_object_init
  Checks and Assertions, 331
__factory_init
  Dynamic Objects Factory, 493
__heap_init
  Memory Heaps, 429
__idle_thread
  OS Instances, 64
__instance_get_currthread
  OS Instances, 63
__instance_set_currthread
  OS Instances, 63
__oslib_init
  Version Numbers and Identification, 348
__reg_object_init
  Registry, 318
__rfcu_object_init
  Runtime Faults Collection Unit, 68
__sch_get_currthread
  Scheduler, 92
__sch_ready_ahead
  Scheduler, 94
__sch_readyBehind
  Scheduler, 92
__sch_reschedule_ahead
  Scheduler, 95
__sch_reschedule_behind
  Scheduler, 95
__stats_ctxswc
  Statistics, 342
__stats_increase_irq
  Statistics, 342
__stats_object_init
  Statistics, 344
__stats_start_measure_crit_isr
Statistics, 344
__stats_start_measure_crit_thd
Statistics, 343
__stats_stop_measure_crit_isr
Statistics, 344
__stats_stop_measure_crit_thd
Statistics, 343
__thd_object_init
Threads, 186
__thd_stackfill
Threads, 187
__tm_calibration_object_init
Time Measurement, 222
__trace_halt
Tracing, 336
__trace_isr_enter
Tracing, 335
__trace_isr_leave
Tracing, 336
__trace_object_init
Tracing, 334
__trace_ready
Tracing, 334
__trace_switch
Tracing, 335
__vt_object_init
Virtual Timers, 175
align
memory_pool_t, 596
ALL_EVENTS
Event Flags, 272
arg
thread_descriptor_t, 603
Base Kernel Services, 105
basemem
memcore_t, 592
best
time_measurement_t, 604
Binary Semaphores, 351
__BSEMAPHORE_DATA, 351
binary_semaphore_t, 352
BSEMAPHORE_DECL, 352
chBSemGetStatel, 360
chBSemObjectInit, 352
chBSemReset, 358
chBSemResetI, 357
chBSemSignal, 359
chBSemSignalI, 359
chBSemWait, 353
chBSemWaitS, 354
chBSemWaitTimeout, 356
chBSemWaitTimeoutS, 355
binary_semaphore_t
Binary Semaphores, 352
BSEMAPHORE_DECL
Binary Semaphores, 352
buf_list
ch_objects_factory, 533
buffer
mailbox_t, 590
pipe_t, 598
trace_buffer_t, 608
cache_sem
ch_objects_cache, 530
ch.h, 615
ch0
System Management, 130
ch1
System Management, 130
ch_binary_semaphore, 511
CH_CFG_CONTEXT_SWITCH_HOOK
Options, 45
CH_CFG_FACTORY_GENERIC_BUFFERS
Dynamic Objects Factory, 490
Options, 39
CH_CFG_FACTORY_MAILBOXES
Dynamic Objects Factory, 491
Options, 40
CH_CFG_FACTORY_MAX_NAMES_LENGTH
Dynamic Objects Factory, 490
Options, 39
CH_CFG_FACTORY_OBJ_FIFOS
Dynamic Objects Factory, 491
Options, 40
CH_CFG_FACTORY_OBJECTS_REGISTRY
Dynamic Objects Factory, 490
Options, 39
CH_CFG_IDLE_ENTER_HOOK
Options, 45
CH_CFG_IDLE_LEAVE_HOOK
Options, 46
CH_CFG_IDLE_LOOP_HOOK
Options, 46
CH_CFG INTERVALS SIZE
Options, 30
CH_CFG_IRQ_EPILOGUE_HOOK
Options, 45
CH_CFG_IRQ_PROLOGUE_HOOK
Options, 45
CH_CFG_MEMCORE_SIZE
Core Memory Manager, 417
Options, 36
CH_CFG_NO_IDLE_THREAD
Options, 31
CH_CFG_OPTIMIZE_SPEED
Options, 31
CH_CFG_OS_INSTANCE_EXTRA_FIELDS
Options, 43
<table>
<thead>
<tr>
<th>Option</th>
<th>Section</th>
</tr>
</thead>
<tbody>
<tr>
<td>CH_CFG_OS_INSTANCE_INIT_HOOK</td>
<td>Options, 43</td>
</tr>
<tr>
<td>CH_CFG_RUNTIME_FAULTS_HOOK</td>
<td>Options, 47</td>
</tr>
<tr>
<td>CH_CFG_SMP_MODE</td>
<td>Options, 29</td>
</tr>
<tr>
<td>CH_CFG_ST_FREQUENCY</td>
<td>Options, 30</td>
</tr>
<tr>
<td>CH_CFG_ST_RESOLUTION</td>
<td>Options, 30</td>
</tr>
<tr>
<td>CH_CFG_ST_TIMEDELTA</td>
<td>Options, 30</td>
</tr>
<tr>
<td>CH_CFG_SYSTEM_EXTRA_FIELDS</td>
<td>Options, 43</td>
</tr>
<tr>
<td>CH_CFG_SYSTEM_HALT_HOOK</td>
<td>Options, 47</td>
</tr>
<tr>
<td>CH_CFG_SYSTEM_INIT_HOOK</td>
<td>Options, 43</td>
</tr>
<tr>
<td>CH_CFG_SYSTEM_TICK_HOOK</td>
<td>Options, 46</td>
</tr>
<tr>
<td>CH_CFG_THREAD_EXIT_HOOK</td>
<td>Options, 44</td>
</tr>
<tr>
<td>CH_CFG_THREAD_EXTRA_FIELDS</td>
<td>Options, 44</td>
</tr>
<tr>
<td>CH_CFG_THREAD_INIT_HOOK</td>
<td>Options, 44</td>
</tr>
<tr>
<td>CH_CFG_TIME_QUANTUM</td>
<td>Options, 31</td>
</tr>
<tr>
<td>CH_CFG_TIME_TYPES_SIZE</td>
<td>Options, 30</td>
</tr>
<tr>
<td>CH_CFG_TRACE_HOOK</td>
<td>Options, 47</td>
</tr>
<tr>
<td>CH_CFG_USE_CONDVARS</td>
<td>Options, 33</td>
</tr>
<tr>
<td>CH_CFG_USE_CONDVARS_TIMEOUT</td>
<td>Options, 34</td>
</tr>
<tr>
<td>CH_CFG_USE_DELEGATES</td>
<td>Options, 38</td>
</tr>
<tr>
<td>CH_CFG_USE_DYNAMIC</td>
<td>Options, 35</td>
</tr>
<tr>
<td>CH_CFG_USE_EVENTS</td>
<td>Options, 34</td>
</tr>
<tr>
<td>CH_CFG_USE_EVENTS_TIMEOUT</td>
<td>Options, 34</td>
</tr>
<tr>
<td>CH_CFG_USE_FACTORY</td>
<td>Options, 39</td>
</tr>
<tr>
<td>CH_CFG_USE_HEAP</td>
<td>Options, 37</td>
</tr>
<tr>
<td>CH_CFG_USE_JOBS</td>
<td>Options, 38</td>
</tr>
<tr>
<td>CH_CFG_USE_MAILBOXES</td>
<td>Options, 36</td>
</tr>
<tr>
<td>CH_CFG_USE_MEMCORE</td>
<td>Options, 36</td>
</tr>
<tr>
<td>CH_CFG_USE_MEMPOOLS</td>
<td>Options, 37</td>
</tr>
<tr>
<td>CH_CFG_USE_MESSAGES</td>
<td>Options, 35</td>
</tr>
<tr>
<td>CH_CFG_USE_MESSAGES_PRIORITY</td>
<td>Options, 35</td>
</tr>
<tr>
<td>CH_CFG_USE_MUTEXES</td>
<td>Options, 33</td>
</tr>
<tr>
<td>CH_CFG_USE_MUTEXES_RECURSIVE</td>
<td>Options, 33</td>
</tr>
<tr>
<td>CH_CFG_USE_OBJ_CACHES</td>
<td>Options, 33</td>
</tr>
<tr>
<td>CH_CFG_USE_OBJ_FIFOS</td>
<td>Options, 37</td>
</tr>
<tr>
<td>CH_CFG_USE_PIPES</td>
<td>Options, 38</td>
</tr>
<tr>
<td>CH_CFG_USE_REGISTRY</td>
<td>Options, 32</td>
</tr>
<tr>
<td>CH_CFG_USE_SEMAPHORES</td>
<td>Options, 32</td>
</tr>
<tr>
<td>CH_CFG_USE_SEMAPHORES_PRIORITY</td>
<td>Options, 33</td>
</tr>
<tr>
<td>CH_DBG_ENABLE_ASSERTS</td>
<td>Options, 41</td>
</tr>
<tr>
<td>CH_DBG_ENABLE_CHECKS</td>
<td>Options, 41</td>
</tr>
<tr>
<td>CH_DBG_ENABLE_STACK_CHECK</td>
<td>Options, 42</td>
</tr>
<tr>
<td>CH_DBG_FILL_THREADS</td>
<td>Options, 42</td>
</tr>
<tr>
<td>CH_DBG_STATISTICS</td>
<td>Options, 40</td>
</tr>
<tr>
<td>CH_DBG_SYSTEM_STATE_CHECK</td>
<td>Options, 40</td>
</tr>
<tr>
<td>CH_DBG_THREADS_PROFILING</td>
<td>Options, 42</td>
</tr>
<tr>
<td>CH_DBG_TRACE_BUFFER_SIZE</td>
<td>Options, 41</td>
</tr>
</tbody>
</table>

ChibiOS/RT
ch_list_t
Lists and Queues, 71
ch_list_unlink
Lists and Queues, 73
ch_memcore
Core Memory Manager, 426
ch_mutex, 526
  cnt, 527
  next, 527
  owner, 527
  queue, 527
ch_objects_cache, 528
  cache_sem, 530
  hashn, 529
  hashp, 529
  lru, 530
  lru_sem, 530
  objn, 529
  objsz, 530
  objvp, 530
  readf, 530
  writef, 531
ch_objects_factory, 531
  buf_list, 533
  fifo_list, 533
  mbx_list, 533
  mtx, 532
  obj_list, 532
  obj_pool, 532
  pipe_list, 533
  sem_list, 533
  sem_pool, 533
ch_objects_fifo, 534
  free, 535
  mbx, 535
ch_oc_hash_header, 535
  hash_next, 536
  hash_prev, 537
ch_oc_lru_header, 537
  hash_next, 538
  hash_prev, 538
  lru_next, 538
  lru_prev, 538
ch_oc_object, 539
  dptr, 541
  hash_next, 540
  hash_prev, 540
  lru_next, 540
  lru_prev, 540
  obj_flags, 541
  obj_group, 540
  obj_key, 541
  obj_sem, 541
ch_os_instance, 542
  config, 544
  core_id, 543
  dbg, 544
  kernel_stats, 544
mainthread, 544
reglist, 543
rfcu, 543
rlist, 543
trace_buffer, 544
vtlist, 543
ch_os_instance_config, 545
  idlethread_base, 546
  idlethread_end, 546
  mainthread_base, 545
  mainthread_end, 546
  name, 545
CH_OSLIB_MAJOR
  Version Numbers and Identification, 348
CH_OSLIB_MINOR
  Version Numbers and Identification, 348
CH_OSLIB_PATCH
  Version Numbers and Identification, 348
CH_OSLIB_STABLE
  Version Numbers and Identification, 347
CH_OSLIB_VERSION
  Version Numbers and Identification, 347
ch_pqueue_init
Lists and Queues, 77
ch_pqueue_insert_ahead
Lists and Queues, 78
ch_pqueue_insert_behind
Lists and Queues, 78
ch_pqueue_remove_highest
Lists and Queues, 77
ch_priority_queue, 546
  next, 547
  prev, 547
  prio, 547
ch_priority_queue_t
Lists and Queues, 71
ch_queue, 548
  next, 548
  prev, 548
CH_QUEUE_DECL
Lists and Queues, 71
ch_queue_dequeue
Lists and Queues, 76
ch_queue_fifo_remove
Lists and Queues, 76
ch_queue_init
Lists and Queues, 75
ch_queue_insert
Lists and Queues, 75
ch_queue_isempty
Lists and Queues, 74
ch_queue_lifo_remove
Lists and Queues, 76
ch_queue_notempty
Lists and Queues, 75
ch_queue_t
Lists and Queues, 71
ch_ready_list, 549
func, 569
par, 569
reload, 569
ch_virtual_timers_list, 570
dist, 571
laststamp, 571
lasttime, 571
systime, 571
chalign.h, 616
chbsem.h, 617
chBSemGetStatel
  Binary Semaphores, 360
chBSemObjectInit
  Binary Semaphores, 352
chBSemReset
  Binary Semaphores, 358
chBSemResetl
  Binary Semaphores, 357
chBSemSignal
  Binary Semaphores, 359
chBSemSignalAll
  Binary Semaphores, 359
chBSemWait
  Binary Semaphores, 353
chBSemWaitS
  Binary Semaphores, 354
chBSemWaitTimeout
  Binary Semaphores, 356
chBSemWaitTimeoutS
  Binary Semaphores, 355
chCacheGetObject
  Objects Caches, 482
chCacheObjectInit
  Objects Caches, 481
chCacheReadObject
  Objects Caches, 483
chCacheReleaseObject
  Objects Caches, 485
chCacheReleaseObjectl
  Objects Caches, 483
chCacheWriteObject
  Objects Caches, 484
chchecks.h, 618
chcond.c, 618
chcond.h, 619
chCondBroadcast
  Condition Variables, 262
chCondBroadcastl
  Condition Variables, 263
chCondObjectInit
  Condition Variables, 259
chCondSignal
  Condition Variables, 260
chCondSignalAll
  Condition Variables, 261
chCondWait
  Condition Variables, 264
chCondWaitS
  Condition Variables, 265
chCondWaitTimeout
  Condition Variables, 266
chCondWaitTimeoutS
  Condition Variables, 267
chconf.h, 620
chCoreAlloc
  Core Memory Manager, 425
chCoreAllocAligned
  Core Memory Manager, 424
chCoreAllocAlignedI
  Core Memory Manager, 423
chCoreAllocAlignedWithOffset
  Core Memory Manager, 418
chCoreAllocAlignedWithOffsetI
  Core Memory Manager, 417
chCoreAllocFromBase
  Core Memory Manager, 421
chCoreAllocFromBaseI
  Core Memory Manager, 419
chCoreAllocFromTop
  Core Memory Manager, 422
chCoreAllocFromTopI
  Core Memory Manager, 420
chCoreAllocI
  Core Memory Manager, 425
chCoreGetStatusX
  Core Memory Manager, 423
chcustomer.h, 623
chDbgAssert
  Checks and Assertions, 324
chDbgCheck
  Checks and Assertions, 324
chDbgCheckClass
  Checks and Assertions, 330
chDbgCheckClassS
  Checks and Assertions, 330
chdebug.c, 624
chdebug.h, 625
chdebug_1, 572
  identifier, 574
  instancesnum, 577
  intctxsize, 577
  intervalsize, 577
  off_ctx, 575
  off_flags, 576
  off_inst_core_id, 579
  off_inst_reglist, 579
  off_inst_rfcu, 579
  off_inst_rlist, 578
  off_inst_rlist_current, 578
  off_inst_vtlist, 578
  off_name, 576
  off_newer, 575
  off_older, 576
  off_preempt, 577
  off_prio, 575
  off_refs, 576
off_state, 576
off_stklimit, 576
off_sys_instances, 578
off_sys_reglist, 578
off_sys_rfcu, 578
off_sys_state, 577
off_time, 577
ptrsize, 575
size, 574
threadsize, 575
timesize, 575
version, 574
zero, 574

cDelegateCallDirect0
Delegate Threads, 394
cDelegateCallDirect1
Delegate Threads, 395
cDelegateCallDirect2
Delegate Threads, 396
cDelegateCallDirect3
Delegate Threads, 397
cDelegateCallDirect4
Delegate Threads, 398
cDelegateCallVeneer
Delegate Threads, 392
cDelegateDispatch
Delegate Threads, 393
cDelegateDispatchTimeout
Delegate Threads, 393
cdelegates.c, 626
cdelegates.h, 627
cdynamic.c, 628
cdynamic.h, 628
cearly.h, 629
Checks, 48
Checks and Assertions, 322
__dbg_check_disable, 325
__dbg_check_enable, 326
__dbg_check_enter_isr, 329
__dbg_check_leave_isr, 329
__dbg_check_lock, 327
__dbg_check_lock_from_isr, 328
__dbg_check_suspend, 326
__dbg_check_unlock, 327
__dbg_check_unlock_from_isr, 328
__dbg_object_init, 331
CH_DBG_STACK_FILL_VALUE, 324
cDbgAssert, 324
cDbgCheck, 324
cDbgCheckClassI, 330
cDbgCheckClassS, 330
system_debug.t, 325
chevents.c, 630
chevents.h, 631
cEvtAddEvents
Event Flags, 278
cEvtAddEventsI
Event Flags, 295
cEvtBroadcast
Event Flags, 294
cEvtBroadcastFlags
Event Flags, 284
cEvtBroadcastFlagsI
Event Flags, 283
cEvtBroadcastI
Event Flags, 294
cEvtDispatch
Event Flags, 285
cEvtGetAndClearEvents
Event Flags, 277
cEvtGetAndClearEventsI
Event Flags, 276
cEvtGetAndClearFlags
Event Flags, 280
cEvtGetAndClearFlagsI
Event Flags, 279
cEvtGetEventsX
Event Flags, 295
cEvtIsListeningI
Event Flags, 293
cEvtObjectInit
Event Flags, 291
cEvtRegister
Event Flags, 292
cEvtRegisterMask
Event Flags, 292
cEvtRegisterMaskWithFlags
Event Flags, 274
cEvtRegisterMaskWithFlagsI
Event Flags, 273
cEvtSignal
Event Flags, 282
cEvtSignalI
Event Flags, 281
cEvtUnregister
Event Flags, 275
cEvtWaitAll
Event Flags, 287
cEvtWaitAllTimeout
Event Flags, 290
cEvtWaitAny
Event Flags, 286
cEvtWaitAnyTimeout
Event Flags, 289
cEvtWaitOne
Event Flags, 285
cEvtWaitOneTimeout
Event Flags, 288
cfactory.c, 633
cfactory.h, 634
chFactoryCreateBuffer
Dynamic Objects Factory, 496
chFactoryCreateMailbox
Dynamic Objects Factory, 500
chFactoryCreateObjectsFIFO
Dynamic Objects Factory, 501
### Memory Heaps
- `chHeapObjectInit` (Memories Heaps, 429)
- `chHeapStatus` (Memory Heaps, 431)

### OS Instances
- `chInstanceObjectInit` (OS Instances, 64)
- `chinstances.c`, 637
- `chinstances.h`, 637

### Jobs Queues
- `chJobDispatch` (Jobs Queues, 412)
- `chJobDispatchTimeout` (Jobs Queues, 413)
- `chJobGet` (Jobs Queues, 402)
- `chJobGetI` (Jobs Queues, 403)
- `chJobGetTimeout` (Jobs Queues, 405)
- `chJobGetTimeoutS` (Jobs Queues, 404)
- `chJobObjectInit` (Jobs Queues, 401)
- `chJobPost` (Jobs Queues, 408)
- `chJobPostAhead` (Jobs Queues, 411)
- `chJobPostAheadI` (Jobs Queues, 409)
- `chJobPostAheadS` (Jobs Queues, 410)
- `chJobPostI` (Jobs Queues, 406)
- `chJobPostS` (Jobs Queues, 407)
- `chjobs.h`, 637
- `chlib.h`, 639
- `chlicense.h`, 640
- `chlists.h`, 641

### Mailboxes
- `chMBFetchI` (Mailboxes, 375)
- `chMBFetchTimeout` (Mailboxes, 373)
- `chMBFetchTimeoutS` (Mailboxes, 374)
- `chMBGetFreeCountI` (Mailboxes, 377)
- `chMBGetSizeI` (Mailboxes, 376)
- `chMBGetUsedCountI` (Mailboxes, 377)
- `chMBObjectInit` (Mailboxes, 364)
- `chmboxes.c`, 643
- `chmboxes.h`, 643

### Synchronous Messages
- `chMBPostAheadTimeout` (Synchronous Messages, 370)
- `chMBPostAheadTimeoutS` (Synchronous Messages, 371)
- `chMBPostI` (Mailboxes, 369)
- `chMBPostTimeout` (Mailboxes, 367)
- `chMBPostTimeoutS` (Mailboxes, 368)
- `chMBReset` (Mailboxes, 365)
- `chMBResetl` (Mailboxes, 366)
- `chMBResumeX` (Mailboxes, 379)

### Mutexes
- `chMtxGetNextMutexX` (Mutexes, 257)
- `chMtxGetOwnerI` (Mutexes, 256)
- `chMtxLock` (Mutexes, 247)
- `chMtxLockS` (Mutexes, 248)
- `chMtxObjectInit` (Mutexes, 246)
chSemResetI
  Counting Semaphores, 239
chSemResetWithMessage
  Counting Semaphores, 228
chSemResetWithMessageI
  Counting Semaphores, 229
chSemSignal
  Counting Semaphores, 234
chSemSignalI
  Counting Semaphores, 235
chSemSignalWait
  Counting Semaphores, 237
chSemWait
  Counting Semaphores, 230
chSemWaitS
  Counting Semaphores, 231
chSemWaitTimeout
  Counting Semaphores, 232
chSemWaitTimeoutS
  Counting Semaphores, 233
chstats.c, 669
chstats.h, 670
chsys.c, 670
chsys.h, 671
chSysDisable
  System Management, 123
chSysEnable
  System Management, 125
chSysGetIdleThreadX
  System Management, 118
chSysGetRealtimeCounterX
  System Management, 115
chSysGetStatusAndLockX
  System Management, 120
chSysHalt
  OS Types and Structures, 61
  System Management, 117
chSysInit
  System Management, 117
chSysIntegrityCheckI
  System Management, 118
chSysIsCounterWithinX
  System Management, 122
chSysLock
  System Management, 125
chSysLockFromISR
  System Management, 126
chSysNotifyInstance
  System Management, 129
chSysPolledDelayX
  System Management, 123
chSysRestoreStatusX
  System Management, 121
chSysSuspend
  System Management, 124
chSysSwitch
  System Management, 115
chSysTimerHandlerI
  System Management, 119
chSysUnconditionalLock
  System Management, 128
chSysUnconditionalUnlock
  System Management, 128
chSysUnlock
  System Management, 126
chSysUnlockFromISR
  System Management, 127
chSysWaitSystemState
  System Management, 116
chThdAddRef
  Threads, 194
chThdCreate
  Threads, 190
chThdCreateFromHeap
  Dynamic Threads, 308
chThdCreateFromMemoryPool
  Dynamic Threads, 309
chThdCreateI
  Threads, 189
chThdCreateStatic
 Threads, 192
chThdCreateSuspended
  Threads, 188
chThdCreateSuspendedI
  Threads, 187
chThdDequeueAllI
  Threads, 212
chThdDequeueNextI
  Threads, 211
chThdDoDequeueNextI
  Threads, 218
chThdEnqueueTimeoutS
  Threads, 210
chThdExit
  Threads, 196
chThdExitS
  Threads, 197
chThdGetPriorityX
  Threads, 213
chThdGetSelfIX
  Threads, 213
chThdGetTicksX
  Threads, 213
chThdGetWorkingAreaX
  Threads, 214
chThdQueueIsEmptyI
  Threads, 217
chThdQueueObjectInit
  Threads, 217
chThdResume
  Threads, 195
chThdResume
  Threads, 209
chThdResumeI
  Threads, 207
chThdResumeS
ChibiOS/RT
...
Event Flags, 270
  __EVENTSOURCE_DATA, 272
  ALL_EVENTS, 272
  chEvtAddEvents, 278
  chEvtAddEventsI, 295
  chEvtBroadcast, 294
  chEvtBroadcastFlags, 284
  chEvtBroadcastFlagsI, 283
  chEvtBroadcastl, 294
  chEvtDispatch, 285
  chEvtGetAndClearEvents, 277
  chEvtGetAndClearEventsI, 276
  chEvtGetAndClearFlags, 280
  chEvtGetAndClearFlagsI, 279
  chEvtGetEventsX, 295
  chEvtIsListeningI, 293
  chEvtObjectInit, 291
  chEvtRegister, 292
  chEvtRegisterMask, 292
  chEvtRegisterMaskWithFlags, 274
  chEvtRegisterMaskWithFlagsI, 273
  chEvtSignal, 282
  chEvtSignall, 281
  chEvtUnregister, 275
  chEvtWaitAll, 287
  chEvtWaitAllTimeout, 290
  chEvtWaitAny, 286
  chEvtWaitAnyTimeout, 289
  chEvtWaitOne, 285
  chEvtWaitOneTimeout, 288
  EVENT_MASK, 272
  event_source_t, 273
  EVENTSOURCE_DECL, 273
  evhandler_t, 273

  event_listener, 580
  events, 581
  flags, 581
  listener, 581
  next, 581
  wflags, 582
  EVENT_MASK
    Event Flags, 272
    event_source, 582
    next, 583
  event_source_t
    Event Flags, 273
  eventflags_t
    OS Types and Structures, 58
  eventid_t
    OS Types and Structures, 58
  eventmask_t
    OS Types and Structures, 58
  events
    event_listener, 581
  EVENTSOURCE_DECL
    Event Flags, 273
  evhandler_t
    Event Flags, 273
  ewmask
    ch_thread, 565
  exitcode
    ch_thread, 564

  FALSE
    Version Numbers and Identification, 24
  fifo
    ch_dyn_objects_fifo, 519
  fifo_list
    ch_objects_factory, 533
  firstprio
    Scheduler, 92
  flags
    ch_thread, 563
    event_listener, 581
  free
    ch_jobs_queue, 524
    ch_objects_fifo, 535
  func
    ch_virtual_timer, 569
  funcp
    thread_descriptor_t, 603
  guarded_memory_pool_t, 584
    pool, 585
    sem, 585
  GUARDEDMEMORYPOOL_DECL
    Memory Pools, 437
  halt
    trace_event_t, 613
  hash_get_s
    Objects Caches, 479
  hash_next
    ch_oc_hash_header, 536
    ch_oc_lru_header, 538
    ch_oc_object, 540
  hash_prev
    ch_oc_hash_header, 537
    ch_oc_lru_header, 538
    ch_oc_object, 540
  hashn
    ch_objects_cache, 529
  hashp
    ch_objects_cache, 529
  hdr
    ch_thread, 561
  header
    memory_heap, 594
  heap
    heap_header, 586
  heap_header
    heap, 586
    next, 586
    pages, 586
    size, 586
  heap_header_t
    Memory Heaps, 429
CH_CFG_OPTIMIZE_SPEED, 31
CH_CFG_OS_INSTANCE_EXTRA_FIELDS, 43
CH_CFG_RUNTIME_FAULTS_HOOK, 47
CH_CFG_SMP_MODE, 29
CH_CFG_ST_FREQUENCY, 30
CH_CFG_ST_RESOLUTION, 29
CH_CFG_ST_TIMEDELTA, 30
CH_CFG_SYSTEM_EXTRA_FIELDS, 43
CH_CFG_SYSTEM_HALT_HOOK, 47
CH_CFG_SYSTEM_INIT_HOOK, 43
CH_CFG_SYSTEM_TICK_HOOK, 46
CH_CFG_THREAD_EXIT_HOOK, 44
CH_CFG_THREAD_EXTRA_FIELDS, 44
CH_CFG_THREAD_INIT_HOOK, 44
CH_CFG_TIME_QUANTUM, 31
CH_CFG_TIME_TYPES_SIZE, 30
CH_CFG_TRACE_HOOK, 47
CH_CFG_USE_CONDVARS, 33
CH_CFG_USE_CONDVARS_TIMEOUT, 34
CH_CFG_USE_DELEGATES, 38
CH_CFG_USE_DYNAMIC, 35
CH_CFG_USE_EVENTS, 34
CH_CFG_USE_EVENTS_TIMEOUT, 34
CH_CFG_USE_FACTORY, 39
CH_CFG_USE_HEAP, 37
CH_CFG_USE_JOBS, 38
CH_CFG_USE_MAILBOXES, 36
CH_CFG_USE_MEMCORE, 36
CH_CFG_USE_MEMPOOLS, 37
CH_CFG_USE_MESSAGES, 35
CH_CFG_USE_MESSAGES_PRIORITY, 35
CH_CFG_USE_MUTEXES, 33
CH_CFG_USE_MUTEXES_RECURSIVE, 33
CH_CFG_USE_OBJ_CACHES, 38
CH_CFG_USE_OBJ_FIFOS, 37
CH_CFG_USE_PIPES, 38
CH_CFG_USE_REGISTRY, 32
CH_CFG_USE_SEMAPHORES, 32
CH_CFG_USE_SEMAPHORES_PRIORITY, 33
CH_CFG_USE_TIMESTAMP, 32
CH_CFG_USE_TM, 31
CH_CFG_USE_WAITEXIT, 32
CH_DBG_ENABLE_ASSERTS, 41
CH_DBG_ENABLE_CHECKS, 41
CH_DBG_ENABLE_STACK_CHECK, 42
CH_DBG_FILL_THREADS, 42
CH_DBG_STATISTICS, 40
CH_DBG_SYSTEM_STATE_CHECK, 40
CH_DBG_THREADS_PROFILING, 42
CH_DBG_TRACE_BUFFER_SIZE, 41
CH_DBG_TRACE_MASK, 41
OS Types and Structures, 52
__CH_OFFSETOF, 55
__CH_STRINGIFY, 53
__CH_USED, 55
ch_system_t, 61
chSysHalt, 61
cnt_t, 58
core_id_t, 59
eventflags_t, 58
eventid_t, 58
eventmask_t, 58
likely, 55
msg_t, 58
os_instance_config_t, 61
os_instance_t, 59
ready_list_t, 61
registry_t, 60
rtcnt_t, 57
rttime_t, 57
stkalign_t, 57
syssts_t, 57
system_state_t, 61
thread_reference_t, 60
thread_t, 59
threadref, 56
threads_queue_t, 60
tmode_t, 57
tprio_t, 58
trefs_t, 57
tslices_t, 58
tstate_t, 57
ucnt_t, 59
unlikely, 56
virtual_timer_t, 59
virtual_timers_list_t, 60
vtfunc_t, 59
os_instance_config_t
OS Types and Structures, 61
os_instance_t
OS Types and Structures, 59
owner
ch_mutex, 527
ch_thread, 562
pages
heap_header, 586
panic_msg
ch_system_debug, 558
par
ch_virtual_timer, 569
pipe
ch_dyn_pipe, 521
PIPE DECL
Pipes, 381
pipe_list
ch_objects_factory, 533
pipe_read
Pipes, 382
pipe_t, 596

ChibiOS/RT
buffer, 598
cnt, 599
dcnt, 599
rdptr, 599
reset, 599
rmtx, 600
rt, 599
top, 598
wmtx, 600
wrptr, 598
wtr, 599
pipe_write
Pipes, 381
Pipes, 380
__PIPE_DATA, 380
chPipeGetFreeCount, 386
chPipeGetSize, 385
chPipeGetUsedCount, 385
chPipeObjectInit, 383
chPipeReadTimeout, 384
chPipeReset, 383
chPipeResume, 386
chPipeWriteTimeout, 383
PIPE_DECL, 381
pipe_read, 382
pipe_write, 381
pool
    guarded_memory_pool_t, 585
pool_header, 600
    next, 601
Port Interface, 51
pqueue
    ch_ready_list, 550
    ch_thread, 561
prev
    ch_delta_list, 513
    ch_priority_queue, 547
    ch_queue, 548
prio
    ch_priority_queue, 547
    thread_descriptor_t, 602
provider
    memory_heap, 594
    memory_pool_t, 596
ptr
    trace_buffer_t, 608
ptrsize
    chdebug_t, 575
qr
    mailbox_t, 591
queue
    ch_mutex, 527
    ch_registry, 552
    ch_semaphore, 555
    ch_thread, 561
    ch_threads_queue, 568
    condition_variable, 580
qw
    mailbox_t, 591
    rdptr
        mailbox_t, 590
        pipe_t, 599
    rdy
        trace_event_t, 612
    rdymsg
        ch_thread, 563
    readf
        ch_objects_cache, 530
    ready_list_t
        OS Types and Structures, 61
    realprio
        ch_thread, 566
    reason
        trace_event_t, 613
    refs
        ch_dyn_element, 514
        ch_thread, 563
REG_HEADER
    Registry, 312
REG_INSERT
    Registry, 313
REG_REMOVE
    Registry, 312
registered_object_t
    Dynamic Objects Factory, 492
Registry, 311
    __reg_object_init, 318
    chRegFindThreadByName, 315
    chRegFindThreadByPointer, 316
    chRegFindThreadByWorkingArea, 317
    chRegFirstThread, 313
    chRegGetThreadNameX, 319
    chRegNextThread, 314
    chRegSetThreadName, 318
    chRegSetThreadNameX, 319
REG_HEADER, 312
REG_INSERT, 313
REG_REMOVE, 312
registry_t
    OS Types and Structures, 60
reglist
    ch_os_instance, 543
    ch_system, 557
Release and Licensing, 15
Release Information, 16
    __CHIBIOS__, 16
    CH_VERSION, 16
    CH_VERSION_DATE, 17
    CH_VERSION_MONTH, 17
    CH_VERSION_NICKNAME, 17
    CH_VERSION_PATCH, 17
    CH_VERSION_STABLE, 16
    CH_VERSION_YEAR, 17
reload
    ch_virtual_timer, 569
    reset
INDEX

mailbox_t, 591
pipe_t, 599
Restrictions, 49
rfcu
    ch_os_instance, 543
ch_system, 557
rfcu_mask_t
    Runtime Faults Collection Unit, 67
rfcu_t
    Runtime Faults Collection Unit, 67
rlist
    ch_os_instance, 543
rmtx
    pipe_t, 600
rqueue
    ch_thread, 562
RT Kernel, 22
RTC2MS
    System Management, 114
RTC2S
    System Management, 113
RTC2US
    System Management, 114
rcnt_t
    OS Types and Structures, 57
rtr
    pipe_t, 599
rtstamp
    trace_event_t, 611
rttime_t
    OS Types and Structures, 57
Runtime Faults Collection Unit, 66
    __rfcu_object_init, 68
    CH_RFCU_ALL_FAULTS, 66
    chRFCUCollectFaultsI, 67
    chRFCUGetAndClearFaultsI, 67
    rfcu_mask_t, 67
    rfcu_t, 67
S2RTC
    System Management, 111
Scheduler, 84
    __sch_get_curthread, 92
    __sch_ready_ahead, 94
    __sch_ready_behind, 92
    __sch_reschedule_ahead, 95
    __sch_reschedule_behind, 95
    CH_FLAG_MODE_HEAP, 91
    CH_FLAG_MODE_MASK, 91
    CH_FLAG_MODE_MPOOL, 91
    CH_FLAG_MODE_STATIC, 91
    CH_FLAG_TERMINATE, 92
    ch_sch_prio_insert, 96
    CH_STATE_CURRENT, 88
    CH_STATE_FINAL, 90
    CH_STATE_NAMES, 91
    CH_STATE_QUEUE, 89
    CH_STATE_READY, 88
    CH_STATE_SLEEPING, 89
    CH_STATE_SNDMSG, 90
    CH_STATE_SNDMSGQ, 90
    CH_STATE_SUSPENDED, 88
    CH_STATE_WTANDEVT, 90
    CH_STATE_WTCREATE, 89
    CH_STATE_WTEXIT, 89
    CH_STATE_WTMEM, 90
    CH_STATE_WTMutex, 89
    CH_STATE_WTREADY, 89
    CH_STATE_WTSTART, 88
    chSchDePreemption, 102
    chSchDoPreemptionRequired, 101
    chSchDePreemption, 103
    chSchDeSleep, 98
    chSchDoSleepTimeout, 99
    chSchIsPreemptionRequired, 101
    chSchPreemption, 97
    CH_FLAG_MODE_HEAP, 91
    CH_FLAG_MODE_MASK, 91
    CH_FLAG_MODE_MPOOL, 91
    CH_FLAG_MODE_STATIC, 91
    CH_FLAG_TERMINATE, 92
    chSchDoPreemption, 102
    chSchDoPreemptionRequired, 101
    chSchDePreemption, 103
    chSchDeSleep, 98
    chSchDoSleepTimeout, 99
    chSchIsPreemptionRequired, 101
    chSchPreemption, 97
    HIGHPRIO, 88
    IDLEPRIO, 87
    LOWPRIO, 87
    MSG_OK, 87
    MSG_RESET, 87
    MSG_TIMEOUT, 87
    NOPRIO, 87
    NORMALPRIO, 88
sem
    ch_dyn_semaphore, 522
    guarded_memory_pool_t, 585
sem_list
    ch_objects_factory, 533
sem_pool
    ch_objects_factory, 533
SEMAPHORE_DECL
    Counting Semaphores, 227
semaphore_t
    Counting Semaphores, 227
sentmsg
    ch_thread, 564
size
    chdebug_t, 574
    heap_header, 586
    trace_buffer_t, 608
state
    ch_system, 556
    ch_thread, 562
    trace_event_t, 611
Statistics, 342
    __stats_ctxswc, 342
    __stats_increase_irq, 342
    __stats_object_init, 344
    __stats_start_measure_crit_isr, 344
    __stats_start_measure_crit_thd, 343
    __stats_stop_measure_crit_isr, 344
__stats_stop_measure_crit_thd, 343
stats
ch_thread, 567
stkalign_t
  OS Types and Structures, 57
suspended
  trace_buffer_t, 608
sw
  trace_event_t, 612
Synchronization, 224, 350
Synchronous Messages, 297
  chMsgGet, 306
  chMsgIsPendingI, 305
  chMsgPoll, 304
  chMsgPollIS, 300
  chMsgRelease, 301
  chMsgReleaseS, 307
  chMsgSend, 298
  chMsgWait, 302
  chMsgWaitIS, 298
  chMsgWaitTimeout, 303
  chMsgWaitTimeoutS, 299
sysinterval_t
  Time and Intervals, 140
systats_t
  OS Types and Structures, 57
System, 50
System Management, 106
  ch0, 130
  ch1, 130
  ch_core0_cfg, 130
  ch_core1_cfg, 130
CH_FAST_IRQ_HANDLER, 111
CH_IRQ_EPILOGUE, 110
CH_IRQ_HANDLER, 111
CH_IRQ_IS_VALID_KERNEL_PRIORITY, 109
CH_IRQ_IS_VALID_PRIORITY, 109
CH_IRQ_PROLOGUE, 110
CH_SYS_CORE0_MEMORY, 108
CH_SYS_CORE1_MEMORY, 108
ch_system, 129
chSysDisable, 123
chSysEnable, 125
chSysGetIdleThreadX, 118
chSysGetRealtimeCounterX, 115
chSysGetStatusAndLockX, 120
chSysHalt, 117
chSysInit, 117
chSysIntegrityCheckI, 118
chSysIsCounterWithinX, 122
chSysLock, 125
chSysLockFromISR, 126
chSysNotifyInstance, 129
chSysPolledDelayX, 123
chSysRestoreStatusX, 121
chSysSuspend, 124
chSysSwitch, 115
chSysTimerHandlerI, 119
chSysUnconditionalLock, 128
chSysUnconditionalUnlock, 128
chSysUnlock, 126
chSysUnlockFromISR, 127
chSysWaitSystemState, 116
currcore, 109
MS2RTC, 112
RTC2MS, 114
RTC2S, 113
RTC2US, 114
S2RTC, 111
THD_WORKING_AREA, 116
US2RTC, 113
system_debug_t
  Checks and Assertions, 325
system_state_t
  OS Types and Structures, 61
systime
  ch_virtual_timers_list, 571
systime_t
  Time and Intervals, 140
systimestamp_t
  Time and Intervals, 140
tfunc_t
  Threads, 186
THD_DESCRIPTOR
  Threads, 182
THD_DESCRIPTOR_AFFINITY
  Threads, 184
THD_FUNCTION
  Threads, 182
THD_WORKING_AREA
  System Management, 116
  Threads, 181
THD_WORKING_AREA_BASE
  Threads, 181
THD_WORKING_AREA_END
  Threads, 182
THD_WORKING_AREA_SIZE
  Threads, 180
thread_descriptor_t, 601
  arg, 603
  funcp, 603
  instance, 603
  name, 602
  prio, 602
  wbase, 602
  wend, 602
thread_reference_t
  OS Types and Structures, 60
thread_t
  OS Types and Structures, 59
threadref
  OS Types and Structures, 56
Threads, 177
  __THREADS_QUEUE_DATA, 180
  __thd_object_init, 186
  __thd_stackfill, 187
chThdAddRef, 194
chThdCreate, 190
chThdCreateStatic, 192
chThdCreateSuspended, 198
chThdCreateSuspendedI, 197
chThdDequeueAllI, 212
chThdDequeueNextI, 211
chThdDoDequeueNextI, 218
chThdEnqueueTimeoutS, 210
chThdExit, 196
chThdExitS, 197
chThdGetPriorityX, 213
chThdGetSelfX, 213
chThdGetTicksX, 213
chThdGetWorkingAreaX, 214
chThdQueueIsEmptyI, 217
chThdQueueObjectInit, 217
chThdRelease, 195
chThdResume, 209
chThdResumeI, 207
chThdResumeS, 208
chThdSetPriority, 199
chThdShouldTerminateX, 215
chThdSleep, 201
chThdSleepMicroseconds, 185
chThdSleepMilliseconds, 185
chThdSleepS, 216
chThdSleepSeconds, 184
chThdSleepUntil, 202
chThdSleepUntilWindowed, 203
chThdStart, 193
chThdStartl, 215
chThdSuspendS, 205
chThdSuspendTimeoutS, 206
chThdTerminate, 200
chThdTerminatedX, 214
chThdWait, 198
chThdYield, 204
tfunc_t, 186
THD_DESCRIPTOR, 182
THD_DESCRIPTOR_AFFINITY, 184
THD_FUNCTION, 182
THD_WORKING_AREA, 181
THD_WORKING_AREA_BASE, 181
THD_WORKING_AREA_END, 182
THD_WORKING_AREA_SIZE, 180
THREADS_QUEUE_DECL, 180
THREADS_QUEUE_DECL
Threads, 180
threads_queue_t
  OS Types and Structures, 60
threadsize
cdebug_t, 575
ticks
ch_thread, 563
time
ch_thread, 563
trace_event_t, 611
Time and Intervals, 134
chTimeAddX, 144
chTimeDiffX, 145
chTimeI2MS, 143
chTimeI2S, 143
chTimeI2US, 144
chTimeInRangeX, 145
chTimeMS2I, 142
chTimeS2I, 141
chTimeStampAddX, 146
chTimeStampDiffX, 146
chTimeStampInRangeX, 147
chTimeUS2I, 142
sysinterval_t, 140
systime_t, 140
systimestamp_t, 140
time_conv_t, 141
TIME_I2MS, 138
TIME_I2S, 138
TIME_I2US, 139
TIME_IMMEDIATE, 135
TIME_INFINITE, 135
TIME_MAX_INTERVAL, 136
TIME_MAX_SYSTIME, 136
TIME_MS2I, 137
time_msecs_t, 141
TIME_S2I, 136
time_secs_t, 140
TIME_US2I, 137
time_usecs_t, 141
Time Measurement, 220
__tm_calibration_object_init, 222
chTMChainMeasurementToX, 222
chTMObjectInit, 220
chTMStartMeasurementX, 221
chTMStopMeasurementX, 221
TM_CALIBRATION_LOOP, 220
time_conv_t
  Time and Intervals, 141
TIME_I2MS
  Time and Intervals, 138
TIME_I2S
  Time and Intervals, 138
TIME_I2US
  Time and Intervals, 139
TIME_IMMEDIATE
  Time and Intervals, 135
TIME_INFINITE
  Time and Intervals, 135
TIME_MAX_INTERVAL
  Time and Intervals, 136
TIME_MAX_SYSTIME
  Time and Intervals, 136
time_measurement_t, 603
  best, 604
cumulative, 605
  last, 604
n, 605
worst, 604

TIME_MS2I
Time and Intervals, 137
time_msecs_t
Time and Intervals, 141

TIME_SI
Time and Intervals, 136
time_secs_t
Time and Intervals, 140

TIME_US2I
Time and Intervals, 137
time_usecs_t
Time and Intervals, 141
timesize
chdebug_t, 575

TM_CALIBRATION_LOOP
Time Measurement, 220
tm_calibration_t, 605
offset, 606
tmc
ch_system, 557
tmode_t
OS Types and Structures, 57
top
mailbox_t, 590
pipe_t, 598
topmem
memcore_t, 592
tp
trace_event_t, 612
tprio_t
OS Types and Structures, 58
trace_buffer
ch_os_instance, 544
trace_buffer_t, 606
buffer, 608
ptr, 608
size, 608
suspended, 608
trace_event_t, 609
halt, 613
isr, 612
msg, 612
name, 612
ntp, 611
rdy, 612
reason, 613
rtstamp, 611
state, 611
sw, 612
time, 611
tp, 612
type, 611
up1, 613
up2, 613
user, 613
wtobjp, 611
trace_next
Tracing, 334
Tracing, 332
__trace_halt, 336
__trace_isr_enter, 335
__trace_isr_leave, 336
__trace_object_init, 334
__trace_ready, 334
__trace_switch, 335
CH_DBG_TRACE_BUFFER_SIZE, 333
CH_DBG_TRACE_MASK, 333
chTraceResume, 340
chTraceResumeI, 340
chTraceSuspend, 339
chTraceSuspendI, 338
chTraceWrite, 337
chTraceWriteI, 336
trace_next, 334
trefs_t
OS Types and Structures, 57
TRUE
Version Numbers and Identification, 24
tslices_t
OS Types and Structures, 58
tstate_t
OS Types and Structures, 57
type
trace_event_t, 611
u
ch_thread, 565
ucnt_t
OS Types and Structures, 59
unlikely
OS Types and Structures, 56
up1
trace_event_t, 613
up2
trace_event_t, 613
US2RTC
System Management, 113
user
trace_event_t, 613
version
chdebug_t, 574
Version Numbers and Identification, 23, 347
__CHIBIOS_OSLIB__, 347
__CHIBIOS_RT__, 23
__oslib_init, 348
CH_KERNEL_MAJOR, 24
CH_KERNEL_MINOR, 24
CH_KERNEL_PATCH, 24
CH_KERNEL_STABLE, 23
CH_KERNEL_VERSION, 24
CH_OSLIB_MAJOR, 348
CH_OSLIB_MINOR, 348
CH_OSLIB_PATCH, 348
CH_OSLIB_STABLE, 347