# ChibiOS/RT

## 1 Copyright

### 1.1 Copyright

### 1.2 Introduction

### 1.3 Related Documents

## 2 Kernel Concepts

### 2.1 Naming Conventions

### 2.2 API Name Suffixes

### 2.3 Interrupt Classes

### 2.4 System States

### 2.5 Scheduling

### 2.6 Thread States

### 2.7 Priority Levels

### 2.8 Thread Working Area

## 3 Module Index

### 3.1 Modules

## 4 Hierarchical Index

### 4.1 Class Hierarchy

## 5 Data Structure Index

### 5.1 Data Structures

## 6 File Index

### 6.1 File List

## 7 Module Documentation

### 7.1 Release and Licensing

#### 7.1.1 Detailed Description

### 7.2 Release Information

#### 7.2.1 Detailed Description

#### 7.2.2 Macro Definition Documentation

##### 7.2.2.1 __CHIBIOS__

##### 7.2.2.2 CH_VERSION_STABLE

##### 7.2.2.3 CH_VERSION

##### 7.2.2.4 CH_VERSION_YEAR

##### 7.2.2.5 CH_VERSION_MONTH

##### 7.2.2.6 CH_VERSION_PATCH

##### 7.2.2.7 CH_VERSION_NICKNAME

##### 7.2.2.8 CH_VERSION_DATE

### 7.3 Customer Information

#### 7.3.1 Detailed Description

#### 7.3.2 Macro Definition Documentation
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.8.2.9</td>
<td>CH_CFG_OPTIMIZE_SPEED</td>
<td>37</td>
</tr>
<tr>
<td>7.8.2.10</td>
<td>CH_CFG_USE_TM</td>
<td>38</td>
</tr>
<tr>
<td>7.8.2.11</td>
<td>CH_CFG_USE_TIMESTAMP</td>
<td>38</td>
</tr>
<tr>
<td>7.8.2.12</td>
<td>CH_CFG_USE_REGISTRY</td>
<td>38</td>
</tr>
<tr>
<td>7.8.2.13</td>
<td>CH_CFG_USE_WAITEXIT</td>
<td>38</td>
</tr>
<tr>
<td>7.8.2.14</td>
<td>CH_CFG_USE_SEMAPHORES</td>
<td>39</td>
</tr>
<tr>
<td>7.8.2.15</td>
<td>CH_CFG_USE_SEMAPHORES_PRIORITY</td>
<td>39</td>
</tr>
<tr>
<td>7.8.2.16</td>
<td>CH_CFG_USE_MUTEXES</td>
<td>39</td>
</tr>
<tr>
<td>7.8.2.17</td>
<td>CH_CFG_USE_MUTEXES_RECURSIVE</td>
<td>39</td>
</tr>
<tr>
<td>7.8.2.18</td>
<td>CH_CFG_USE_CONDVARS</td>
<td>40</td>
</tr>
<tr>
<td>7.8.2.19</td>
<td>CH_CFG_USE_CONDVARS_TIMEOUT</td>
<td>40</td>
</tr>
<tr>
<td>7.8.2.20</td>
<td>CH_CFG_USE_EVENTS</td>
<td>40</td>
</tr>
<tr>
<td>7.8.2.21</td>
<td>CH_CFG_USE_EVENTS_TIMEOUT</td>
<td>41</td>
</tr>
<tr>
<td>7.8.2.22</td>
<td>CH_CFG_USE_MESSAGES</td>
<td>41</td>
</tr>
<tr>
<td>7.8.2.23</td>
<td>CH_CFG_USE_MESSAGES_PRIORITY</td>
<td>41</td>
</tr>
<tr>
<td>7.8.2.24</td>
<td>CH_CFG_USE_DYNAMIC</td>
<td>42</td>
</tr>
<tr>
<td>7.8.2.25</td>
<td>CH_CFG_USE_MAILBOXES</td>
<td>42</td>
</tr>
<tr>
<td>7.8.2.26</td>
<td>CH_CFG_USE_MEMCORE</td>
<td>42</td>
</tr>
<tr>
<td>7.8.2.27</td>
<td>CH_CFG_MEMCORE_SIZE</td>
<td>43</td>
</tr>
<tr>
<td>7.8.2.28</td>
<td>CH_CFG_USE_HEAP</td>
<td>43</td>
</tr>
<tr>
<td>7.8.2.29</td>
<td>CH_CFG_USE_MEMPOOLS</td>
<td>43</td>
</tr>
<tr>
<td>7.8.2.30</td>
<td>CH_CFG_USE_OBJ_FIFOS</td>
<td>44</td>
</tr>
<tr>
<td>7.8.2.31</td>
<td>CH_CFG_USE_PIPES</td>
<td>44</td>
</tr>
<tr>
<td>7.8.2.32</td>
<td>CH_CFG_USE_OBJ_CACHES</td>
<td>44</td>
</tr>
<tr>
<td>7.8.2.33</td>
<td>CH_CFG_USE_DELEGATES</td>
<td>44</td>
</tr>
<tr>
<td>7.8.2.34</td>
<td>CH_CFG_USE_JOBS</td>
<td>45</td>
</tr>
<tr>
<td>7.8.2.35</td>
<td>CH_CFG_USE_FACTORY</td>
<td>45</td>
</tr>
<tr>
<td>7.8.2.36</td>
<td>CH_CFG_FACTORY_MAX_NAMES_LENGTH</td>
<td>45</td>
</tr>
<tr>
<td>7.8.2.37</td>
<td>CH_CFG_FACTORY_OBJECTS_REGISTRY</td>
<td>45</td>
</tr>
<tr>
<td>7.8.2.38</td>
<td>CH_CFG_FACTORY_GENERIC_BUFFERS</td>
<td>45</td>
</tr>
<tr>
<td>7.8.2.39</td>
<td>CH_CFG_FACTORY_SEMAPHORES</td>
<td>46</td>
</tr>
<tr>
<td>7.8.2.40</td>
<td>CH_CFG_FACTORY_MAILBOXES</td>
<td>46</td>
</tr>
<tr>
<td>7.8.2.41</td>
<td>CH_CFG_FACTORY_OBJ_FIFOS</td>
<td>46</td>
</tr>
<tr>
<td>7.8.2.42</td>
<td>CH_CFG_FACTORY_PIPES</td>
<td>46</td>
</tr>
<tr>
<td>7.8.2.43</td>
<td>CH_DBG_STATISTICS</td>
<td>46</td>
</tr>
<tr>
<td>7.8.2.44</td>
<td>CH_DBG_SYSTEM_STATE_CHECK</td>
<td>47</td>
</tr>
<tr>
<td>7.8.2.45</td>
<td>CH_DBG_ENABLE_CHECKS</td>
<td>47</td>
</tr>
<tr>
<td>7.8.2.46</td>
<td>CH_DBG_ENABLE_ASSERTS</td>
<td>47</td>
</tr>
<tr>
<td>7.8.2.47</td>
<td>CH_DBG_TRACE_MASK</td>
<td>47</td>
</tr>
<tr>
<td>7.8.2.48</td>
<td>CH_DBG_TRACE_BUFFER_SIZE</td>
<td>48</td>
</tr>
<tr>
<td>7.8.2.49</td>
<td>CH_DBG_ENABLE_STACK_CHECK</td>
<td>48</td>
</tr>
<tr>
<td>7.8.2.50</td>
<td>CH_DBG_FILL_THREADS</td>
<td>48</td>
</tr>
</tbody>
</table>
7.8.2.51 CH_DBG_THREADS_PROFILING .................................................. 49
7.8.2.52 CH_CFG_SYSTEM_EXTRA_FIELDS ............................................... 49
7.8.2.53 CH_CFG_SYSTEM_INIT_HOOK .................................................... 49
7.8.2.54 CH_CFG_OS_INSTANCE_EXTRA_FIELDS ....................................... 49
7.8.2.55 CH_CFG_OS_INSTANCE_INIT_HOOK ........................................... 49
7.8.2.56 CH_CFG_THREAD_EXTRA_FIELDS .............................................. 50
7.8.2.57 CH_CFG_THREAD_INIT_HOOK ................................................... 50
7.8.2.58 CH_CFG_THREAD_EXIT_HOOK .................................................. 50
7.8.2.59 CH_CFG_CONTEXT_SWITCH_HOOK ............................................ 51
7.8.2.60 CH_CFG_IRQ_PROLOGUE_HOOK .............................................. 51
7.8.2.61 CH_CFG_IRQ_EPILOGUE_HOOK ............................................... 51
7.8.2.62 CH_CFG_IDLE_ENTER_HOOK ................................................... 52
7.8.2.63 CH_CFG_IDLE_LEAVE_HOOK .................................................. 52
7.8.2.64 CH_CFG_IDLE_LOOP_HOOK .................................................... 52
7.8.2.65 CH_CFG_SYSTEM_TICK_HOOK ................................................ 53
7.8.2.66 CH_CFG_SYSTEM_HALT_HOOK ............................................... 53
7.8.2.67 CH_CFG_TRACE_HOOK ........................................................ 53
7.8.2.68 CH_CFG_RUNTIME_FAULTS_HOOK ......................................... 53

7.9 Checks ................................................................. 54
7.10 Restrictions .......................................................... 55
  7.10.1 Detailed Description ..................................................... 55
7.11 System ............................................................... 56
  7.11.1 Detailed Description ..................................................... 56
7.12 Port Interface ........................................................ 57
  7.12.1 Detailed Description ..................................................... 57
7.13 OS Types and Structures .............................................. 58
  7.13.1 Detailed Description ..................................................... 58
  7.13.2 Macro Definition Documentation ....................................... 59
    7.13.2.1 __CH_STRINGIFY ...................................................... 59
    7.13.2.2 __CH_OFFSETOF ...................................................... 60
    7.13.2.3 __CH_USED .......................................................... 60
    7.13.2.4 likely ............................................................... 60
    7.13.2.5 unlikely ............................................................. 61
  7.13.3 Typedef Documentation .............................................. 61
    7.13.3.1 rtcnt_t ............................................................. 61
    7.13.3.2 rttime_t ............................................................. 61
    7.13.3.3 syssts_t ............................................................. 61
    7.13.3.4 stkalign_t ........................................................... 62
    7.13.3.5 tmode_t ............................................................... 62
    7.13.3.6 tstate_t ............................................................... 62
    7.13.3.7 trefs_t ............................................................... 62
    7.13.3.8 tslices_t ............................................................. 62

ChibiOS/RT
7.13.3.9 tprio_t ......................................................... 62
7.13.3.10 msg_t ......................................................... 62
7.13.3.11 eventid_t ..................................................... 63
7.13.3.12 eventmask_t ................................................. 63
7.13.3.13 eventflags_t ............................................... 63
7.13.3.14 cnt_t ......................................................... 63
7.13.3.15 ucnt_t ....................................................... 63
7.13.3.16 core_id_t ..................................................... 63
7.13.3.17 thread_t ....................................................... 64
7.13.3.18 os_instance_t ............................................... 64
7.13.3.19 virtual_timer_t ............................................. 64
7.13.3.20 vfunc_t ....................................................... 64
7.13.3.21 virtual_timers_list_t .................................... 64
7.13.3.22 registry_t ................................................... 65
7.13.3.23 thread_reference_t ....................................... 65
7.13.3.24 threads_queue_t ........................................... 65
7.13.3.25 ready_list_t ............................................... 65
7.13.3.26 os_instance_config_t ..................................... 65
7.13.3.27 ch_system_t ................................................ 66

7.13.4 Enumeration Type Documentation ................................ 66
7.13.4.1 system_state_t .............................................. 66

7.13.5 Function Documentation ........................................ 66
7.13.5.1 chSysHalt() .................................................... 66

7.14 OS Instances ......................................................... 68
7.14.1 Detailed Description ........................................... 68
7.14.2 Macro Definition Documentation ................................ 68
7.14.2.1 __instance_get_currthread ................................ 68
7.14.2.2 __instance_set_currthread ................................ 68

7.14.3 Function Documentation ........................................ 69
7.14.3.1 __idle_thread() .............................................. 69
7.14.3.2 chInstanceObjectInit() .................................... 69

7.15 Runtime Faults Collection Unit ................................... 71
7.15.1 Detailed Description ........................................... 71
7.15.2 Macro Definition Documentation ................................ 71
7.15.2.1 CH_RFCU_ALL_FAULTS ..................................... 71

7.15.3 Typedef Documentation ......................................... 72
7.15.3.1 rfcu_mask_t ................................................... 72
7.15.3.2 rfcu_t ........................................................ 72

7.15.4 Function Documentation ......................................... 72
7.15.4.1 chRFCUCollectFaultsI() ................................... 72
7.15.4.2 chRFCUGetAndClearFaultsI() ................................ 72
7.15.4.3 __rfcu_object_init() ....................................... 73
7.16 Lists and Queues

7.16.1 Detailed Description

7.16.2 Macro Definition Documentation

7.16.2.1 __CH_QUEUE_DATA

7.16.2.2 CH_QUEUEDECL

7.16.3 Typedef Documentation

7.16.3.1 ch_list_t

7.16.3.2 ch_queue_t

7.16.3.3 ch_priority_queue_t

7.16.3.4 ch_delta_list_t

7.16.4 Function Documentation

7.16.4.1 ch_list_init()

7.16.4.2 ch_list_isempty()

7.16.4.3 ch_list_notempty()

7.16.4.4 ch_list_link()

7.16.4.5 ch_list_unlink()

7.16.4.6 ch_queue_init()

7.16.4.7 ch_queue_isempty()

7.16.4.8 ch_queue_notempty()

7.16.4.9 ch_queue_insert()

7.16.4.10 ch_queue_fifo_remove()

7.16.4.11 ch_queue_lifo_remove()

7.16.4.12 ch_queue_dequeue()

7.16.4.13 ch_pqueue_init()

7.16.4.14 ch_pqueue_remove_highest()

7.16.4.15 ch_pqueue_insert_behind()

7.16.4.16 ch_pqueue_insert_ahead()

7.16.4.17 ch_dlist_init()

7.16.4.18 ch_dlist_isempty()

7.16.4.19 ch_dlist_notempty()

7.16.4.20 ch_dlist_islast()

7.16.4.21 ch_dlist_isfirst()

7.16.4.22 ch_dlist_insert_after()

7.16.4.23 ch_dlist_insert_before()

7.16.4.24 ch_dlist_insert()

7.16.4.25 ch_dlist_remove_first()

7.16.4.26 ch_dlist_dequeue()
7.17.2.3 MSG_RESET .................................................. 92
7.17.2.4 NOPRIO .................................................... 92
7.17.2.5 IDLEPRIO .................................................... 92
7.17.2.6 LOWPRIO .................................................... 93
7.17.2.7 NORMALPRIO ................................................. 93
7.17.2.8 HIGHPRIO .................................................... 93
7.17.2.9 CH_STATE_READY ........................................... 93
7.17.2.10 CH_STATE_CURRENT ...................................... 93
7.17.2.11 CH_STATE_WTSTART ..................................... 93
7.17.2.12 CH_STATE_SUSPENDED ................................... 94
7.17.2.13 CH_STATE_QUEUE .......................................... 94
7.17.2.14 CH_STATE_WTSEM ......................................... 94
7.17.2.15 CH_STATE_WTMTX ......................................... 94
7.17.2.16 CH_STATE_WTCOUNT ................................. 94
7.17.2.17 CH_STATE_SLEEPING ..................................... 94
7.17.2.18 CH_STATE_WTEXIT ....................................... 95
7.17.2.19 CH_STATE_WTOREVT .................................... 95
7.17.2.20 CH_STATE_WTANDEV .................................... 95
7.17.2.21 CH_STATE_WTNMSGQ .................................... 95
7.17.2.22 CH_STATE_WDSMSG ....................................... 95
7.17.2.23 CH_STATE_WTMSG ......................................... 95
7.17.2.24 CH_STATE_FINAL .......................................... 96
7.17.2.25 CH_STATE_NAMES .......................................... 96
7.17.2.26 CH_FLAG_MODE_MASK .................................... 96
7.17.2.27 CH_FLAG_MODE_STATIC ................................ 96
7.17.2.28 CH_FLAG_MODE_HEAP .................................... 96
7.17.2.29 CH_FLAG_MODE_MPOOL ................................. 97
7.17.2.30 CH_FLAG_TERMINATE ..................................... 97
7.17.2.31 firstprio .................................................. 97
7.17.2.32 __sch_get_curthread .................................... 97

7.17.3 Function Documentation ........................................ 97
7.17.3.1 __sch_ready_behind() .................................... 98
7.17.3.2 __sch_ready_ahead() ...................................... 99
7.17.3.3 __sch_reschedule_behind() ............................ 100
7.17.3.4 __sch_reschedule_ahead() .............................. 101
7.17.3.5 ch_sch_prio_insert() ..................................... 101
7.17.3.6 chSchReady() .............................................. 102
7.17.3.7 chSchGoSleep() ........................................... 103
7.17.3.8 chSchGoSleepTimeout() .................................. 104
7.17.3.9 chSchWakeup() ............................................. 105
7.17.3.10 chSchReschedule() ...................................... 106
7.17.3.11 chSchIsPreemptionRequired() ........................ 107
<table>
<thead>
<tr>
<th>Section</th>
<th>Function Documentation</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.21.4</td>
<td>chTimeS2l()</td>
<td>146</td>
</tr>
<tr>
<td></td>
<td>chTimeMS2l()</td>
<td>147</td>
</tr>
<tr>
<td></td>
<td>chTimeUS2l()</td>
<td>147</td>
</tr>
<tr>
<td></td>
<td>chTime2S()</td>
<td>148</td>
</tr>
<tr>
<td></td>
<td>chTime2MS()</td>
<td>148</td>
</tr>
<tr>
<td></td>
<td>chTime2US()</td>
<td>149</td>
</tr>
<tr>
<td></td>
<td>chTimeAddX()</td>
<td>149</td>
</tr>
<tr>
<td></td>
<td>chTimeDiffX()</td>
<td>150</td>
</tr>
<tr>
<td></td>
<td>chTimeIsInRangeX()</td>
<td>150</td>
</tr>
<tr>
<td></td>
<td>chTimeStampAddX()</td>
<td>151</td>
</tr>
<tr>
<td></td>
<td>chTimeStampDiffX()</td>
<td>151</td>
</tr>
<tr>
<td></td>
<td>chTimeStampIsInRangeX()</td>
<td>152</td>
</tr>
<tr>
<td>7.22</td>
<td>Virtual Timers</td>
<td>153</td>
</tr>
<tr>
<td>7.22.1</td>
<td>Detailed Description</td>
<td>153</td>
</tr>
<tr>
<td>7.22.2</td>
<td>Function Documentation</td>
<td>154</td>
</tr>
<tr>
<td></td>
<td>vt_insert_first()</td>
<td>154</td>
</tr>
<tr>
<td></td>
<td>vt_enqueue()</td>
<td>155</td>
</tr>
<tr>
<td></td>
<td>chVTDoSetI()</td>
<td>155</td>
</tr>
<tr>
<td></td>
<td>chVTDoSetContinuousI()</td>
<td>156</td>
</tr>
<tr>
<td></td>
<td>chVTDoResetI()</td>
<td>157</td>
</tr>
<tr>
<td></td>
<td>chVTGetRemainingIntervalI()</td>
<td>158</td>
</tr>
<tr>
<td></td>
<td>chVTDoTickI()</td>
<td>159</td>
</tr>
<tr>
<td></td>
<td>chVTGetTimeStampI()</td>
<td>160</td>
</tr>
<tr>
<td></td>
<td>chVTResetTimeStampI()</td>
<td>161</td>
</tr>
<tr>
<td></td>
<td>chVTObjectInit()</td>
<td>162</td>
</tr>
<tr>
<td></td>
<td>chVTGetSystemTimeX()</td>
<td>162</td>
</tr>
<tr>
<td></td>
<td>chVTGetSystemTime()</td>
<td>163</td>
</tr>
<tr>
<td></td>
<td>chVTTimeElapsedSinceX()</td>
<td>163</td>
</tr>
<tr>
<td></td>
<td>chVTIsSystemTimeWithinX()</td>
<td>164</td>
</tr>
<tr>
<td></td>
<td>chVTIsSystemTimeWithin()</td>
<td>165</td>
</tr>
<tr>
<td></td>
<td>chVTGetTimersStateI()</td>
<td>166</td>
</tr>
<tr>
<td></td>
<td>chVTIsArmedI()</td>
<td>167</td>
</tr>
<tr>
<td></td>
<td>chVTIsArmed()</td>
<td>168</td>
</tr>
<tr>
<td></td>
<td>chVTRestSet()</td>
<td>169</td>
</tr>
<tr>
<td></td>
<td>chVTResetI()</td>
<td>170</td>
</tr>
<tr>
<td></td>
<td>chVTSetI()</td>
<td>171</td>
</tr>
<tr>
<td></td>
<td>chVTSet()</td>
<td>172</td>
</tr>
<tr>
<td></td>
<td>chVTSetContinuous()</td>
<td>173</td>
</tr>
<tr>
<td></td>
<td>chVTSetContinuousI()</td>
<td>174</td>
</tr>
<tr>
<td></td>
<td>chVTGetReloadIntervalX()</td>
<td>175</td>
</tr>
<tr>
<td></td>
<td>chVTGetReloadInterval()</td>
<td>176</td>
</tr>
</tbody>
</table>
7.22.2.27 chVTGetTimeStamp() ................................................. 176
7.22.2.28 chVTResetTimeStamp() ............................................. 177
7.22.2.29 __vt_object_init() .................................................. 178

7.23 Threads .............................................................................. 180
7.23.1 Detailed Description ......................................................... 180
7.23.2 Macro Definition Documentation ...................................... 183
  7.23.2.1 __THREADS_QUEUE_DATA ......................................... 183
  7.23.2.2 THREADS_QUEUE_DECL ........................................... 183
  7.23.2.3 THD_WORKING_AREA_SIZE ...................................... 184
  7.23.2.4 THD_WORKING_AREA ................................................. 184
  7.23.2.5 THD_WORKING_AREA_BASE ....................................... 184
  7.23.2.6 THD_WORKING_AREA_END ......................................... 185
  7.23.2.7 THD_FUNCTION ..................................................... 185
  7.23.2.8 THD_DESCRIPTOR .................................................. 185
  7.23.2.9 THD_DESCRIPTOR_AFFINITY ...................................... 187
  7.23.2.10 chThdSleepSeconds ............................................... 187
  7.23.2.11 chThdSleepMilliseconds ........................................ 188
  7.23.2.12 chThdSleepMicroseconds ........................................ 188

7.23.3 Typedef Documentation .................................................. 189
  7.23.3.1 tfunc_t ................................................................ 189

7.23.4 Function Documentation .................................................. 189
  7.23.4.1 __thd_object_init() ............................................... 189
  7.23.4.2 __thd_memfill() .................................................... 190
  7.23.4.3 chThdCreateSuspendedI() ....................................... 191
  7.23.4.4 chThdCreateSuspended() ........................................ 192
  7.23.4.5 chThdCreateI() .................................................... 193
  7.23.4.6 chThdCreate() ..................................................... 194
  7.23.4.7 chThdCreateStatic() ............................................. 195
  7.23.4.8 chThdStart() ........................................................ 196
  7.23.4.9 chThdAddRef() ..................................................... 197
  7.23.4.10 chThdRelease() .................................................. 198
  7.23.4.11 chThdExit() ........................................................ 199
  7.23.4.12 chThdExitS() ...................................................... 200
  7.23.4.13 chThdWait() ....................................................... 201
  7.23.4.14 chThdSetPriority() ............................................... 202
  7.23.4.15 chThdTerminate() ............................................... 203
  7.23.4.16 chThdSleep() ...................................................... 204
  7.23.4.17 chThdSleepUntil() ............................................... 205
  7.23.4.18 chThdSleepUntilWindowed() .................................... 206
  7.23.4.19 chThdYield() ....................................................... 207
  7.23.4.20 chThdSuspendS() .................................................. 208
  7.23.4.21 chThdSuspendTimeoutS() ...................................... 209
7.23.4.22 chThdResumeI() ................................................. 210
7.23.4.23 chThdResumeS() ................................................. 211
7.23.4.24 chThdResume() .................................................. 212
7.23.4.25 chThdEnqueueTimeoutS() ....................................... 213
7.23.4.26 chThdDequeueNextI() ........................................... 214
7.23.4.27 chThdDequeueAllI() .............................................. 215
7.23.4.28 chThdGetSelfX() ................................................ 216
7.23.4.29 chThdGetPriorityX() ............................................. 216
7.23.4.30 chThdGetTicksX() ................................................. 216
7.23.4.31 chThdGetWorkingAreaX() ....................................... 217
7.23.4.32 chThdTerminatedX() .............................................. 217
7.23.4.33 chThdShouldTerminateX() ...................................... 218
7.23.4.34 chThdStartI() .................................................... 218
7.23.4.35 chThdSleepS() .................................................... 219
7.23.4.36 chThdQueueObjectInit() ....................................... 220
7.23.4.37 chThdQueueIsEmptyI() ......................................... 220
7.23.4.38 chThdDoDequeueNextI() ....................................... 221

7.24 Time Measurement ....................................................... 223
7.24.1 Detailed Description ................................................ 223
7.24.2 Macro Definition Documentation ................................... 223
7.24.2.1 TM_CALIBRATION_LOOP ........................................ 223
7.24.3 Function Documentation ............................................ 223
7.24.3.1 chTMObjectInit() ................................................. 223
7.24.3.2 chTMStartMeasurementX() ...................................... 224
7.24.3.3 chTMStopMeasurementX() ....................................... 224
7.24.3.4 chTMChainMeasurementToX() ................................... 225
7.24.3.5 __tm_calibration_object_init() ................................. 225

7.25 Synchronization .......................................................... 227
7.25.1 Detailed Description ................................................ 227

7.26 Counting Semaphores ..................................................... 228
7.26.1 Detailed Description ................................................ 228
7.26.2 Macro Definition Documentation ................................... 229
7.26.2.1 __SEMAPHORE_DATA ........................................... 229
7.26.2.2 SEMAPHORE_DECL .............................................. 230
7.26.3 Typedef Documentation ............................................. 230
7.26.3.1 semaphore_t ..................................................... 230
7.26.4 Function Documentation ............................................ 230
7.26.4.1 chSemObjectInit() ................................................. 230
7.26.4.2 chSemResetWithMessage() ...................................... 231
7.26.4.3 chSemResetWithMessageI() ..................................... 232
7.26.4.4 chSemWait() ....................................................... 233
7.26.4.5 chSemWaitS() ..................................................... 234
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.26.4.6 chSemWaitTimeout()</td>
<td>235</td>
</tr>
<tr>
<td>7.26.4.7 chSemWaitTimeoutS()</td>
<td>236</td>
</tr>
<tr>
<td>7.26.4.8 chSemSignal()</td>
<td>237</td>
</tr>
<tr>
<td>7.26.4.9 chSemSignalI()</td>
<td>238</td>
</tr>
<tr>
<td>7.26.4.10 chSemAddCounterI()</td>
<td>239</td>
</tr>
<tr>
<td>7.26.4.11 chSemSignalWait()</td>
<td>240</td>
</tr>
<tr>
<td>7.26.4.12 chSemReset()</td>
<td>241</td>
</tr>
<tr>
<td>7.26.4.13 chSemResetI()</td>
<td>242</td>
</tr>
<tr>
<td>7.26.4.14 chSemFastWait()</td>
<td>243</td>
</tr>
<tr>
<td>7.26.4.15 chSemFastSignalI()</td>
<td>244</td>
</tr>
<tr>
<td>7.26.4.16 chSemGetCounterI()</td>
<td>244</td>
</tr>
<tr>
<td>7.27 Mutexes</td>
<td>246</td>
</tr>
<tr>
<td>7.27.1 Detailed Description</td>
<td>246</td>
</tr>
<tr>
<td>7.27.2 Macro Definition Documentation</td>
<td>247</td>
</tr>
<tr>
<td>7.27.2.1 __MUTEX_DATA</td>
<td>248</td>
</tr>
<tr>
<td>7.27.2.2 MUTEX_DECL</td>
<td>249</td>
</tr>
<tr>
<td>7.27.3 Typedef Documentation</td>
<td>249</td>
</tr>
<tr>
<td>7.27.3.1 mutex_t</td>
<td>249</td>
</tr>
<tr>
<td>7.27.4 Function Documentation</td>
<td>249</td>
</tr>
<tr>
<td>7.27.4.1 chMtxObjectInit()</td>
<td>249</td>
</tr>
<tr>
<td>7.27.4.2 chMtxLock()</td>
<td>250</td>
</tr>
<tr>
<td>7.27.4.3 chMtxLockS()</td>
<td>251</td>
</tr>
<tr>
<td>7.27.4.4 chMtxTryLock()</td>
<td>252</td>
</tr>
<tr>
<td>7.27.4.5 chMtxTryLockS()</td>
<td>253</td>
</tr>
<tr>
<td>7.27.4.6 chMtxUnlock()</td>
<td>254</td>
</tr>
<tr>
<td>7.27.4.7 chMtxUnlockS()</td>
<td>255</td>
</tr>
<tr>
<td>7.27.4.8 chMtxUnlockAllS()</td>
<td>256</td>
</tr>
<tr>
<td>7.27.4.9 chMtxUnlockAll()</td>
<td>257</td>
</tr>
<tr>
<td>7.27.4.10 chMtxQueueNotEmptyS()</td>
<td>258</td>
</tr>
<tr>
<td>7.27.4.11 chMtxGetOwnerI()</td>
<td>259</td>
</tr>
<tr>
<td>7.27.4.12 chMtxGetNextMutexX()</td>
<td>260</td>
</tr>
<tr>
<td>7.28 Condition Variables</td>
<td>261</td>
</tr>
<tr>
<td>7.28.1 Detailed Description</td>
<td>261</td>
</tr>
<tr>
<td>7.28.2 Macro Definition Documentation</td>
<td>262</td>
</tr>
<tr>
<td>7.28.2.1 __CONDVAR_DATA</td>
<td>262</td>
</tr>
<tr>
<td>7.28.2.2 CONDVARDECL</td>
<td>262</td>
</tr>
<tr>
<td>7.28.3 Typedef Documentation</td>
<td>262</td>
</tr>
<tr>
<td>7.28.3.1 condition_variable_t</td>
<td>262</td>
</tr>
<tr>
<td>7.28.4 Function Documentation</td>
<td>262</td>
</tr>
<tr>
<td>7.28.4.1 chCondObjectInit()</td>
<td>262</td>
</tr>
<tr>
<td>7.28.4.2 chCondSignal()</td>
<td>263</td>
</tr>
<tr>
<td>7.28.4.3 chCondSignalI()</td>
<td>264</td>
</tr>
</tbody>
</table>
7.28.4.4 chCondBroadcast() .......................................................... 265
7.28.4.5 chCondBroadcastI() ......................................................... 266
7.28.4.6 chCondWait() ................................................................. 267
7.28.4.7 chCondWaitS() ............................................................... 268
7.28.4.8 chCondWaitTimeout() ..................................................... 269
7.28.4.9 chCondWaitTimeoutS() .................................................... 271

7.29 Event Flags ................................................................. 273

7.29.1 Detailed Description .................................................... 273

7.29.2 Macro Definition Documentation .................................. 275
7.29.2.1 ALL EVENTS ................................................................. 275
7.29.2.2 EVENT_MASK .............................................................. 275
7.29.2.3 __EVENTSOURCE_DATA ................................................ 275
7.29.2.4 EVENTSOURCE_DECL .................................................. 276

7.29.3 Typedef Documentation ................................................ 276
7.29.3.1 event_source_t ........................................................... 276
7.29.3.2 evhandler_t ............................................................... 276

7.29.4 Function Documentation .............................................. 276
7.29.4.1 chEvtRegisterMaskWithFlagsI() .................................... 277
7.29.4.2 chEvtRegisterMaskWithFlags() ..................................... 277
7.29.4.3 chEvtUnregister() .......................................................... 278
7.29.4.4 chEvtGetAndClearEventsI() .......................................... 279
7.29.4.5 chEvtGetAndClearEvents() ........................................... 280
7.29.4.6 chEvtAddEvents() ....................................................... 281
7.29.4.7 chEvtGetAndClearFlagsI() ........................................... 281
7.29.4.8 chEvtGetAndClearFlags() .............................................. 282
7.29.4.9 chEvtSignalI() ............................................................ 283
7.29.4.10 chEvtSignal() ............................................................ 284
7.29.4.11 chEvtBroadcastFlagsI() .............................................. 285
7.29.4.12 chEvtBroadcastFlags() ................................................. 286
7.29.4.13 chEvtDispatch() .......................................................... 287
7.29.4.14 chEvtWaitOne() ......................................................... 288
7.29.4.15 chEvtWaitAny() .......................................................... 289
7.29.4.16 chEvtWaitAll() ........................................................... 290
7.29.4.17 chEvtWaitOneTimeout() .............................................. 291
7.29.4.18 chEvtWaitAnyTimeout() .............................................. 292
7.29.4.19 chEvtWaitAllTimeout() .............................................. 293
7.29.4.20 chEvtObjectInit() ....................................................... 294
7.29.4.21 chEvtRegisterMask() .................................................. 295
7.29.4.22 chEvtRegister() .......................................................... 296
7.29.4.23 chEvtIsListeningI() ..................................................... 297
7.29.4.24 chEvtBroadcast() ........................................................ 297
7.29.4.25 chEvtBroadcastI() ...................................................... 298
7.29.4.26 chEvtAddEventsI() .......................................................... 298
7.29.4.27 chEvtGetEventsX() ......................................................... 298

7.30 Synchronous Messages ......................................................... 300
7.30.1 Detailed Description ......................................................... 300
7.30.2 Function Documentation .................................................... 301
  7.30.2.1 chMsgSend() ................................................................. 301
  7.30.2.2 chMsgWaitS() ............................................................... 301
  7.30.2.3 chMsgWaitTimeoutS() .................................................... 302
  7.30.2.4 chMsgPollS() ............................................................... 303
  7.30.2.5 chMsgRelease() ........................................................... 304
  7.30.2.6 chMsgWait() ............................................................... 305
  7.30.2.7 chMsgWaitTimeout() ..................................................... 306
  7.30.2.8 chMsgPoll() ............................................................... 307
  7.30.2.9 chMsgIsPendingI() ....................................................... 309
  7.30.2.10 chMsgGet() ............................................................... 309
  7.30.2.11 chMsgReleaseS() ....................................................... 310

7.31 Dynamic Threads .................................................................. 311
7.31.1 Detailed Description ......................................................... 311
7.31.2 Function Documentation .................................................... 311
  7.31.2.1 chThdCreateFromHeap() ................................................. 311
  7.31.2.2 chThdCreateFromMemoryPool() ....................................... 312

7.32 Registry .............................................................................. 314
7.32.1 Detailed Description ......................................................... 314
7.32.2 Macro Definition Documentation ......................................... 315
  7.32.2.1 REG_HEADER .................................................................. 315
  7.32.2.2 REG_REMOVE .................................................................. 315
  7.32.2.3 REG_INSERT ................................................................... 316
7.32.3 Function Documentation .................................................... 316
  7.32.3.1 chRegFirstThread() ......................................................... 316
  7.32.3.2 chRegNextThread() ......................................................... 317
  7.32.3.3 chRegFindThreadByName() ............................................. 318
  7.32.3.4 chRegFindThreadByPointer() ......................................... 319
  7.32.3.5 chRegFindThreadByWorkingArea() ................................. 320
  7.32.3.6 __reg_object_init() ......................................................... 321
  7.32.3.7 chRegSetThreadName() ................................................... 322
  7.32.3.8 chRegGetThreadNameX() ................................................ 322
  7.32.3.9 chRegSetThreadNameX() ................................................ 323

7.33 Debug ................................................................................. 324
7.33.1 Detailed Description ......................................................... 324

7.34 Checks and Assertions ......................................................... 325
7.34.1 Detailed Description ......................................................... 325
7.34.2 Macro Definition Documentation ......................................... 327
7.34.2.1 CH_DBG_STACK_FILL_VALUE ........................................... 327
7.34.2.2 chDbgCheck ......................................................... 327
7.34.2.3 chDbgAssert ......................................................... 327
7.34.3 Typedef Documentation .............................................. 328
7.34.3.1 system_debug_t ..................................................... 328
7.34.4 Function Documentation ............................................ 328
7.34.4.1 __dbg_check_disable() ............................................ 329
7.34.4.2 __dbg_check_suspend() ........................................... 329
7.34.4.3 __dbg_check_enable() ............................................. 330
7.34.4.4 __dbg_check_lock() ................................................ 330
7.34.4.5 __dbg_check_unlock() ............................................. 331
7.34.4.6 __dbg_check_lock_from_isr() ..................................... 331
7.34.4.7 __dbg_check_unlock_from_isr() ................................... 332
7.34.4.8 __dbg_check_enter_isr() .......................................... 332
7.34.4.9 __dbg_check_leave_isr() ......................................... 333
7.34.4.10 chDbgCheckClassI() .............................................. 333
7.34.4.11 chDbgCheckClassS() .............................................. 334
7.34.4.12 __dbg_object_init() ................................................ 334
7.35 Tracing ................................................................. 335
7.35.1 Detailed Description ................................................ 335
7.35.2 Macro Definition Documentation .................................. 336
7.35.2.1 CH_DBG_TRACE_MASK ............................................. 336
7.35.2.2 CH_DBG_TRACE_BUFFER_SIZE .................................... 336
7.35.3 Function Documentation ............................................ 337
7.35.3.1 trace_next() ....................................................... 337
7.35.3.2 __trace_object_init() ............................................. 337
7.35.3.3 __trace_ready() ..................................................... 338
7.35.3.4 __trace_switch() ................................................... 338
7.35.3.5 __trace_isr_enter() ............................................... 338
7.35.3.6 __trace_isr_leave() ............................................... 339
7.35.3.7 __trace_halt() ...................................................... 339
7.35.3.8 chTraceWriteI() ................................................... 339
7.35.3.9 chTraceWrite() ..................................................... 340
7.35.3.10 chTraceSuspendI() ................................................ 340
7.35.3.11 chTraceSuspend() ................................................ 341
7.35.3.12 chTraceResumeI() ................................................ 342
7.35.3.13 chTraceResume() .................................................. 343
7.36 Statistics ............................................................. 345
7.36.1 Detailed Description ................................................ 345
7.36.2 Function Documentation ............................................ 345
7.36.2.1 __stats_increase_irq() .......................................... 345
7.36.2.2 __stats_ctxswc() ................................................. 345
7.36.2.3 __stats_start_measure_crit_thd() ................................................................. 346
7.36.2.4 __stats_stop_measure_crit_thd() ................................................................. 346
7.36.2.5 __stats_start_measure_crit_isr() ................................................................. 347
7.36.2.6 __stats_stop_measure_crit_isr() ................................................................. 347
7.36.2.7 __stats_object_init() ....................................................................................... 348

7.37 OS Library ........................................................................................................... 349

7.37.1 Detailed Description ........................................................................................ 349

7.38 Version Numbers and Identification .................................................................... 350

7.38.1 Detailed Description ........................................................................................ 350

7.38.2 Macro Definition Documentation .................................................................... 350

7.38.2.1 __CHIBIOS_OSLIB__ ......................................................................................... 350
7.38.2.2 CH_OSLIB_STABLE ......................................................................................... 350
7.38.2.3 CH_OSLIB_VERSION ......................................................................................... 351
7.38.2.4 CH_OSLIB_MAJOR ........................................................................................... 351
7.38.2.5 CH_OSLIB_MINOR ........................................................................................... 351
7.38.2.6 CH_OSLIB_PATCH ........................................................................................... 351

7.38.3 Function Documentation .................................................................................. 351

7.38.3.1 __oslib_init() ................................................................................................. 352

7.39 Synchronization ................................................................................................... 353

7.39.1 Detailed Description ......................................................................................... 353

7.40 Binary Semaphores .............................................................................................. 354

7.40.1 Detailed Description ......................................................................................... 354

7.40.2 Macro Definition Documentation .................................................................... 354

7.40.2.1 __BSEMAPHORE_DATA ................................................................................. 355
7.40.2.2 BSEMAPHORE_DECL .................................................................................... 355

7.40.3 Typedef Documentation .................................................................................... 355

7.40.3.1 binary_semaphore_t ......................................................................................... 355

7.40.4 Function Documentation .................................................................................. 355

7.40.4.1 chBSemObjectInit() ......................................................................................... 356
7.40.4.2 chBSemWait() ................................................................................................. 356
7.40.4.3 chBSemWaitS() ............................................................................................... 357
7.40.4.4 chBSemWaitTimeoutS() .................................................................................. 358
7.40.4.5 chBSemWaitTimeout() ...................................................................................... 359
7.40.4.6 chBSemResetI() .............................................................................................. 360
7.40.4.7 chBSemReset() ............................................................................................... 361
7.40.4.8 chBSemSignalI() ............................................................................................ 362
7.40.4.9 chBSemSignal() .............................................................................................. 362
7.40.4.10 chBSemGetStateI() ......................................................................................... 363

7.41 Mailboxes ............................................................................................................. 365

7.41.1 Detailed Description ......................................................................................... 365

7.41.2 Macro Definition Documentation .................................................................... 366

7.41.2.1 __MAILBOX_DATA ......................................................................................... 367
7.41.2.2 MAILBOX_DECL ........................................ 367
7.41.3 Function Documentation .................................. 367
  7.41.3.1 chMBOBJECTInit() .................................. 368
  7.41.3.2 chMBReset() ......................................... 368
  7.41.3.3 chMBResetI() ........................................ 369
  7.41.3.4 chMBPostTimeout() .................................. 370
  7.41.3.5 chMBPostTimeoutS() ................................ 371
  7.41.3.6 chMBPost() .......................................... 372
  7.41.3.7 chMBPostAheadTimeout() ............................. 373
  7.41.3.8 chMBPostAheadTimeoutS() ........................... 374
  7.41.3.9 chMBPostAheadI() .................................. 375
  7.41.3.10 chMBFetchTimeout() ................................ 376
  7.41.3.11 chMBFetchTimeoutS() ................................ 377
  7.41.3.12 chMBFetchI() ...................................... 378
  7.41.3.13 chMBGetSizeI() .................................... 379
  7.41.3.14 chMBGetUsedCountI() ................................ 380
  7.41.3.15 chMBGetFreeCountI() ................................ 380
  7.41.3.16 chMBpeekI() ....................................... 381
  7.41.3.17 chMBResumeX() ..................................... 382

7.42 Pipes ......................................................... 383
  7.42.1 Detailed Description .................................... 383
  7.42.2 Macro Definition Documentation ......................... 383
    7.42.2.1 __PIPE_DATA ...................................... 384
    7.42.2.2 PIPE_DECL ......................................... 384
  7.42.3 Function Documentation ................................ 384
    7.42.3.1 pipe_write() ...................................... 385
    7.42.3.2 pipe_read() ....................................... 385
    7.42.3.3 chPipeObjectInit() ................................ 386
    7.42.3.4 chPipeReset() ..................................... 386
    7.42.3.5 chPipeWriteTimeout() ............................... 387
    7.42.3.6 chPipeReadTimeout() ................................ 387
    7.42.3.7 chPipeGetSize() .................................... 388
    7.42.3.8 chPipeGetUsedCount() ............................. 388
    7.42.3.9 chPipeGetFreeCount() ............................... 389
    7.42.3.10 chPipeResume() .................................... 389

7.43 Delegate Threads ............................................. 391
  7.43.1 Detailed Description .................................... 391
  7.43.2 Typedef Documentation .................................. 392
    7.43.2.1 delegate_veneer_t .................................. 392
    7.43.2.2 delegate_fn0_t ..................................... 392
    7.43.2.3 delegate_fn1_t ..................................... 392
    7.43.2.4 delegate_fn2_t ..................................... 392
7.43.3 Function Documentation

7.43.3.1 __ch_delegate_fn0() .................................................. 392
7.43.3.2 __ch_delegate_fn1() .................................................. 393
7.43.3.3 __ch_delegate_fn2() .................................................. 393
7.43.3.4 __ch_delegate_fn3() .................................................. 393
7.43.3.5 __ch_delegate_fn4() .................................................. 395
7.43.3.6 chDelegateCallVeneer() ........................................... 395
7.43.3.7 chDelegateDispatch() ............................................. 396
7.43.3.8 chDelegateDispatchTimeout() .................................... 397
7.43.3.9 chDelegateCallDirect0() .......................................... 398
7.43.3.10 chDelegateCallDirect1() ......................................... 398
7.43.3.11 chDelegateCallDirect2() ......................................... 399
7.43.3.12 chDelegateCallDirect3() ......................................... 400
7.43.3.13 chDelegateCallDirect4() ......................................... 401

7.44 Jobs Queues .................................................................... 403

7.44.1 Detailed Description ..................................................... 403
7.44.2 Macro Definition Documentation ................................... 404
7.44.2.1 MSG_JOB_NULL ...................................................... 404
7.44.3 Typedef Documentation ................................................ 404
7.44.3.1 jobs_queue_t ......................................................... 404
7.44.3.2 job_function_t ....................................................... 404
7.44.3.3 job_descriptor_t ..................................................... 404
7.44.4 Function Documentation .............................................. 404
7.44.4.1 chJobObjectInit() .................................................. 404
7.44.4.2 chJobGet() .......................................................... 405
7.44.4.3 chJobGetI() .......................................................... 406
7.44.4.4 chJobGetTimeoutS() .............................................. 407
7.44.4.5 chJobGetTimeout() ................................................ 408
7.44.4.6 chJobPostI() ......................................................... 409
7.44.4.7 chJobPostS() ........................................................ 410
7.44.4.8 chJobPost() ........................................................ 411
7.44.4.9 chJobPostAheadI() ............................................... 412
7.44.4.10 chJobPostAheadS() ............................................. 413
7.44.4.11 chJobPostAhead() ............................................... 414
7.44.4.12 chJobDispatch() ................................................... 415
7.44.4.13 chJobDispatchTimeout() ...................................... 416

7.45 Memory Management ..................................................... 418
7.45.1 Detailed Description ................................................... 418

7.46 Core Memory Manager .................................................... 419
7.46.1 Detailed Description ................................................... 419
7.50.3.10 chFifoSendObjectS() .............................................. 472
7.50.3.11 chFifoSendObject() ............................................. 473
7.50.3.12 chFifoSendObjectAheadI() ...................................... 474
7.50.3.13 chFifoSendObjectAheadS() ...................................... 475
7.50.3.14 chFifoSendObjectAhead() ....................................... 476
7.50.3.15 chFifoReceiveObject() .......................................... 476
7.50.3.16 chFifoReceiveObjectTimeoutS() ................................ 477
7.50.3.17 chFifoReceiveObjectTimeout() .................................. 478

7.51 Objects Caches ......................................................... 480
7.51.1 Detailed Description ................................................. 480
7.51.2 Typedef Documentation ............................................. 481
  7.51.2.1 oc_flags_t ........................................................... 481
  7.51.2.2 oc_hash_header_t .................................................. 481
  7.51.2.3 oc_lru_header_t .................................................... 481
  7.51.2.4 oc_object_t ........................................................ 482
  7.51.2.5 objects_cache_t ................................................... 482
  7.51.2.6 oc_readf_t .......................................................... 482
  7.51.2.7 oc_writef_t .......................................................... 482
7.51.3 Function Documentation ........................................... 482
  7.51.3.1 hash_get_s() ....................................................... 483
  7.51.3.2 lru_get_last_s() .................................................. 483
  7.51.3.3 chCacheObjectInit() ............................................. 484
  7.51.3.4 chCacheGetObject() ............................................. 485
  7.51.3.5 chCacheReleaseObjectI() ...................................... 486
  7.51.3.6 chCacheReadObject() ........................................... 487
  7.51.3.7 chCacheWriteObject() .......................................... 487
  7.51.3.8 chCacheReleaseObject() ....................................... 488

7.52 Dynamic Objects Factory ............................................. 490
7.52.1 Detailed Description ................................................. 490
7.52.2 Macro Definition Documentation ................................ 493
  7.52.2.1 CH_CFG_FACTORY_MAX_NAMES_LENGTH ........................... 493
  7.52.2.2 CH_CFG_FACTORY_OBJECTS_REGISTRY ......................... 493
  7.52.2.3 CH_CFG_FACTORY_GENERIC BUFFERS .............................. 493
  7.52.2.4 CH_CFG_FACTORY_SEMAPHORES [1/2] .............................. 493
  7.52.2.5 CH_CFG_FACTORY_SEMAPHORES [2/2] .............................. 494
  7.52.2.6 CH_CFG_FACTORY_MAILBOXES [1/2] .............................. 494
  7.52.2.7 CH_CFG_FACTORY_MAILBOXES [2/2] .............................. 494
  7.52.2.8 CH_CFG_FACTORY_OBJ_FIFOS [1/3] .............................. 494
  7.52.2.9 CH_CFG_FACTORY_OBJ_FIFOS [2/3] .............................. 494
  7.52.2.10 CH_CFG_FACTORY_OBJ_FIFOS [3/3] ............................. 494
  7.52.2.11 CH_CFG_FACTORY_PIPES [1/2] ................................ 495
  7.52.2.12 CH_CFG_FACTORY_PIPES [2/2] ................................ 495
8 Data Structure Documentation

8.1 ch_binary_semaphore Struct Reference
   8.1.1 Detailed Description

8.2 ch_delta_list Struct Reference
   8.2.1 Detailed Description
   8.2.2 Field Documentation
      8.2.2.1 next
      8.2.2.2 prev
      8.2.2.3 delta

8.3 ch_dyn_element Struct Reference
   8.3.1 Detailed Description
   8.3.2 Field Documentation
      8.3.2.1 next
      8.3.2.2 refs

8.4 ch_dyn_list Struct Reference
   8.4.1 Detailed Description

8.5 ch_dyn_mailbox Struct Reference
   8.5.1 Detailed Description
   8.5.2 Field Documentation
      8.5.2.1 element
      8.5.2.2 mbx

8.6 ch_dyn_object Struct Reference
   8.6.1 Detailed Description
   8.6.2 Field Documentation
      8.6.2.1 element

8.7 ch_dyn_objects_fifo Struct Reference
   8.7.1 Detailed Description
   8.7.2 Field Documentation
      8.7.2.1 element
      8.7.2.2 fifo

8.8 ch_dyn_pipe Struct Reference
   8.8.1 Detailed Description
   8.8.2 Field Documentation
      8.8.2.1 element
      8.8.2.2 pipe

8.9 ch_dyn_semaphore Struct Reference
   8.9.1 Detailed Description
   8.9.2 Field Documentation
      8.9.2.1 element
      8.9.2.2 sem

8.10 ch_job_descriptor Struct Reference
    8.10.1 Detailed Description
8.28.2.1 queue .......................................................... 559
8.28.2.2 cnt .......................................................... 559

8.29 ch_system Struct Reference .................................. 559
  8.29.1 Detailed Description ...................................... 560
  8.29.2 Field Documentation ...................................... 560
    8.29.2.1 state .................................................. 561
    8.29.2.2 instances ............................................. 561
    8.29.2.3 tmc .................................................... 561
    8.29.2.4 reglist ............................................... 561
    8.29.2.5 rfcu .................................................. 561

8.30 ch_system_debug Struct Reference .......................... 562
  8.30.1 Detailed Description ...................................... 562
  8.30.2 Field Documentation ...................................... 562
    8.30.2.1 panic_msg ............................................ 562
    8.30.2.2 isr_cnt .............................................. 563
    8.30.2.3 lock_cnt ............................................. 563

8.31 ch_thread Struct Reference .................................. 563
  8.31.1 Detailed Description ...................................... 565
  8.31.2 Field Documentation ...................................... 565
    8.31.2.1 list .................................................. 565
    8.31.2.2 queue ................................................. 565
    8.31.2.3 pqueue ............................................... 565
    8.31.2.4 hdr .................................................. 566
    8.31.2.5 ctx .................................................. 566
    8.31.2.6 rqueue ............................................... 566
    8.31.2.7 owner ................................................. 566
    8.31.2.8 name ................................................ 566
    8.31.2.9 wabase .............................................. 566
    8.31.2.10 state ................................................. 567
    8.31.2.11 flags ............................................... 567
    8.31.2.12 refs ................................................ 567
    8.31.2.13 ticks ............................................... 567
    8.31.2.14 time ................................................. 567
    8.31.2.15 rdymsg .............................................. 568
    8.31.2.16 exitcode ............................................ 568
    8.31.2.17 wtoinp ............................................. 568
    8.31.2.18 wttrp ............................................... 568
    8.31.2.19 sentmsg ............................................. 568
    8.31.2.20 wtsemp ............................................. 569
    8.31.2.21 wtmtxp .............................................. 569
    8.31.2.22 ewmask .............................................. 569
    8.31.2.23 u .................................................... 570
8.31.2.24 waiting ......................................................... 570
8.31.2.25 msgqueue .................................................. 570
8.31.2.26 epending .................................................... 570
8.31.2.27 mtxlist ..................................................... 570
8.31.2.28 realprio .................................................... 571
8.31.2.29 mpool ....................................................... 571
8.31.2.30 stats ....................................................... 571

8.32 ch_threads_queue Struct Reference ........................................... 571
8.32.1 Detailed Description ............................................... 572
8.32.2 Field Documentation ............................................... 572
  8.32.2.1 queue ................................................... 572

8.33 ch_virtual_timer Struct Reference ........................................... 572
8.33.1 Detailed Description ............................................... 573
8.33.2 Field Documentation ............................................... 573
  8.33.2.1 dlist ................................................... 573
  8.33.2.2 func ................................................... 573
  8.33.2.3 par ................................................... 573
  8.33.2.4 reload .................................................. 573

8.34 ch_virtual_timers_list Struct Reference .................................... 574
8.34.1 Detailed Description ............................................... 574
8.34.2 Field Documentation ............................................... 575
  8.34.2.1 dlist ................................................... 575
  8.34.2.2 systime .................................................. 575
  8.34.2.3 lasttime ................................................ 575
  8.34.2.4 laststamp ................................................ 575

8.35 chdebug_t Struct Reference ............................................... 576
8.35.1 Detailed Description ............................................... 577
8.35.2 Field Documentation ............................................... 577
  8.35.2.1 identifier ............................................... 577
  8.35.2.2 zero .................................................... 577
  8.35.2.3 size .................................................... 578
  8.35.2.4 version .................................................. 578
  8.35.2.5 ptrsize .................................................. 578
  8.35.2.6 timesize ................................................ 578
  8.35.2.7 threadsize .............................................. 578
  8.35.2.8 off_prio ................................................ 578
  8.35.2.9 off_ctx ................................................ 579
  8.35.2.10 off_newer .............................................. 579
  8.35.2.11 off_older .............................................. 579
  8.35.2.12 off_name .............................................. 579
  8.35.2.13 off_stklimit ......................................... 579
  8.35.2.14 off_state .............................................. 579
8.35.2.15 off_flags .......................................................... 580
8.35.2.16 off.refs ....................................................... 580
8.35.2.17 off.preempt .................................................. 580
8.35.2.18 off.time ...................................................... 580

8.36 condition_variable Struct Reference ........................................ 580
8.36.1 Detailed Description ............................................... 581
8.36.2 Field Documentation ............................................... 581
8.36.2.1 queue .......................................................... 581

8.37 event_listener Struct Reference ......................................... 581
8.37.1 Detailed Description ............................................... 582
8.37.2 Field Documentation ............................................... 582
8.37.2.1 next .......................................................... 582
8.37.2.2 listener ....................................................... 582
8.37.2.3 events ......................................................... 582
8.37.2.4 flags ........................................................ 583
8.37.2.5 wflags ........................................................ 583

8.38 event_source Struct Reference .......................................... 583
8.38.1 Detailed Description ............................................... 584
8.38.2 Field Documentation ............................................... 584
8.38.2.1 next .......................................................... 585

8.39 guarded_memory_pool_t Struct Reference .................................. 585
8.39.1 Detailed Description ............................................... 586
8.39.2 Field Documentation ............................................... 586
8.39.2.1 sem .......................................................... 586
8.39.2.2 pool .......................................................... 586

8.40 heap_header Union Reference ............................................ 586
8.40.1 Detailed Description ............................................... 587
8.40.2 Field Documentation ............................................... 587
8.40.2.1 next .......................................................... 587
8.40.2.2 pages ......................................................... 587
8.40.2.3 heap ........................................................ 587
8.40.2.4 size .......................................................... 587

8.41 kernel_stats_t Struct Reference ........................................ 588
8.41.1 Detailed Description ............................................... 588
8.41.2 Field Documentation ............................................... 589
8.41.2.1 n_irq ......................................................... 589
8.41.2.2 n.cctxswc .................................................. 589
8.41.2.3 m.crit.thd .................................................. 589
8.41.2.4 m.crit.isr .................................................. 589

8.42 mailbox_t Struct Reference ............................................. 590
8.42.1 Detailed Description ............................................... 591
8.42.2 Field Documentation ............................................... 591
8.42.2.1 buffer .............................. 591
8.42.2.2 top ................................ 591
8.42.2.3 wrptr ............................. 591
8.42.2.4 rdptr ............................. 592
8.42.2.5 cnt ............................... 592
8.42.2.6 reset ............................. 592
8.42.2.7 qw ............................... 592
8.42.2.8 qr ............................... 592

8.43 memcore_t Struct Reference .............................. 593
  8.43.1 Detailed Description .......................... 593
  8.43.2 Field Documentation .......................... 593
    8.43.2.1 basemem ............................ 593
    8.43.2.2 topmem ............................ 593

8.44 memory_heap Struct Reference ........................... 594
  8.44.1 Detailed Description .......................... 595
  8.44.2 Field Documentation .......................... 595
    8.44.2.1 provider ............................ 595
    8.44.2.2 header .............................. 595
    8.44.2.3 mtx ............................... 595

8.45 memory_pool_t Struct Reference .......................... 596
  8.45.1 Detailed Description .......................... 596
  8.45.2 Field Documentation .......................... 596
    8.45.2.1 next .............................. 597
    8.45.2.2 object_size ......................... 597
    8.45.2.3 align .............................. 597
    8.45.2.4 provider ............................ 597

8.46 pipe_t Struct Reference .................................. 597
  8.46.1 Detailed Description .......................... 599
  8.46.2 Field Documentation .......................... 599
    8.46.2.1 buffer ............................ 599
    8.46.2.2 top ............................... 599
    8.46.2.3 wrptr ............................. 600
    8.46.2.4 rdptr ............................. 600
    8.46.2.5 cnt ............................... 600
    8.46.2.6 reset ............................. 600
    8.46.2.7 wtr ............................... 600
    8.46.2.8 rtr ............................... 600
    8.46.2.9 cmtx .............................. 601
    8.46.2.10 wmtx ............................. 601
    8.46.2.11 rmtx ............................. 601

8.47 pool_header Struct Reference ............................ 601
  8.47.1 Detailed Description .......................... 602
8.47.2 Field Documentation .................................................. 602
  8.47.2.1 next ................................................................. 602

8.48 thread_descriptor_t Struct Reference ................................ 602
  8.48.1 Detailed Description .............................................. 603
  8.48.2 Field Documentation .............................................. 603
    8.48.2.1 name ............................................................. 603
    8.48.2.2 wbase ............................................................ 603
    8.48.2.3 wend ............................................................. 603
    8.48.2.4 prio .............................................................. 604
    8.48.2.5 funcp ............................................................ 604
    8.48.2.6 arg .............................................................. 604
    8.48.2.7 instance ......................................................... 604

8.49 time_measurement_t Struct Reference .................................. 604
  8.49.1 Detailed Description .............................................. 605
  8.49.2 Field Documentation .............................................. 605
    8.49.2.1 best .............................................................. 605
    8.49.2.2 worst ............................................................. 605
    8.49.2.3 last ............................................................... 606
    8.49.2.4 n ................................................................. 606
    8.49.2.5 cumulative ....................................................... 606

8.50 tm_calibration_t Struct Reference .................................... 606
  8.50.1 Detailed Description .............................................. 607
  8.50.2 Field Documentation .............................................. 607
    8.50.2.1 offset ............................................................ 607

8.51 trace_buffer_t Struct Reference ..................................... 607
  8.51.1 Detailed Description .............................................. 609
  8.51.2 Field Documentation .............................................. 609
    8.51.2.1 suspended ....................................................... 609
    8.51.2.2 size .............................................................. 609
    8.51.2.3 ptr .............................................................. 609
    8.51.2.4 buffer ........................................................... 609

8.52 trace_event_t Struct Reference ..................................... 610
  8.52.1 Detailed Description .............................................. 611
  8.52.2 Field Documentation .............................................. 612
    8.52.2.1 type .............................................................. 612
    8.52.2.2 state ............................................................ 612
    8.52.2.3 rtstamp .......................................................... 612
    8.52.2.4 time ............................................................ 612
    8.52.2.5 ntp .............................................................. 612
    8.52.2.6 wtobjp ........................................................... 613
    8.52.2.7 sw .............................................................. 613
    8.52.2.8 tp .............................................................. 613
9 File Documentation

9.1 ch.h File Reference
9.1.1 Detailed Description
9.2 chalign.h File Reference
9.2.1 Detailed Description
9.3 chbsem.h File Reference
9.3.1 Detailed Description
9.4 chchecks.h File Reference
9.4.1 Detailed Description
9.5 chcond.c File Reference
9.5.1 Detailed Description
9.6 chcond.h File Reference
9.6.1 Detailed Description
9.7 chconf.h File Reference
9.7.1 Detailed Description
9.8 chcustomer.h File Reference
9.8.1 Detailed Description
9.9 chdebug.c File Reference
9.9.1 Detailed Description
9.10 chdebug.h File Reference
9.10.1 Detailed Description
9.11 chdelegates.c File Reference
9.11.1 Detailed Description
9.12 chdelegates.h File Reference
9.12.1 Detailed Description
9.13 chdynamic.c File Reference
9.13.1 Detailed Description
9.14 chdynamic.h File Reference
9.14.1 Detailed Description
9.15 chearly.h File Reference
9.15.1 Detailed Description
9.16 chevents.c File Reference
9.37.1 Detailed Description .................................................. 653
9.38 chobjcaches.c File Reference ......................................... 653
  9.38.1 Detailed Description .......................................................... 654
9.39 chobjcaches.h File Reference ........................................... 654
  9.39.1 Detailed Description .......................................................... 656
9.40 chobjects.h File Reference .............................................. 656
  9.40.1 Detailed Description .......................................................... 657
9.41 chobjfifos.h File Reference ............................................. 657
  9.41.1 Detailed Description .......................................................... 658
9.42 chpipes.c File Reference ................................................ 658
  9.42.1 Detailed Description .......................................................... 659
9.43 chpipes.h File Reference ................................................. 659
  9.43.1 Detailed Description .......................................................... 660
9.44 chport.h File Reference .................................................. 660
  9.44.1 Detailed Description .......................................................... 660
9.45 chregistry.c File Reference ............................................. 660
  9.45.1 Detailed Description .......................................................... 661
9.46 chregistry.h File Reference ............................................. 661
  9.46.1 Detailed Description .......................................................... 662
9.47 chrestrictions.h File Reference ....................................... 662
  9.47.1 Detailed Description .......................................................... 662
9.48 chrfcu.c File Reference .................................................. 662
  9.48.1 Detailed Description .......................................................... 663
9.49 chrfcu.h File Reference .................................................. 663
  9.49.1 Detailed Description .......................................................... 663
9.50 chschd.c File Reference .................................................. 664
  9.50.1 Detailed Description .......................................................... 664
9.51 chschd.h File Reference .................................................. 664
  9.51.1 Detailed Description .......................................................... 667
9.52 chsem.c File Reference ................................................. 667
  9.52.1 Detailed Description .......................................................... 668
9.53 chsem.h File Reference ................................................. 668
  9.53.1 Detailed Description .......................................................... 669
9.54 chstats.c File Reference ............................................... 669
  9.54.1 Detailed Description .......................................................... 669
9.55 chstats.h File Reference ............................................... 670
  9.55.1 Detailed Description .......................................................... 670
9.56 chsys.c File Reference .................................................. 670
  9.56.1 Detailed Description .......................................................... 671
9.57 chsys.h File Reference .................................................. 671
  9.57.1 Detailed Description .......................................................... 673
9.58 chtthreads.c File Reference .......................................... 673
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>9.58.1</td>
<td>Detailed Description</td>
<td>675</td>
</tr>
<tr>
<td>9.59</td>
<td>chthreads.h File Reference</td>
<td>675</td>
</tr>
<tr>
<td>9.59.1</td>
<td>Detailed Description</td>
<td>677</td>
</tr>
<tr>
<td>9.60</td>
<td>ctime.h File Reference</td>
<td>678</td>
</tr>
<tr>
<td>9.60.1</td>
<td>Detailed Description</td>
<td>679</td>
</tr>
<tr>
<td>9.61</td>
<td>chtm.c File Reference</td>
<td>679</td>
</tr>
<tr>
<td>9.61.1</td>
<td>Detailed Description</td>
<td>680</td>
</tr>
<tr>
<td>9.62</td>
<td>chtm.h File Reference</td>
<td>680</td>
</tr>
<tr>
<td>9.62.1</td>
<td>Detailed Description</td>
<td>680</td>
</tr>
<tr>
<td>9.63</td>
<td>chtrace.c File Reference</td>
<td>680</td>
</tr>
<tr>
<td>9.63.1</td>
<td>Detailed Description</td>
<td>680</td>
</tr>
<tr>
<td>9.64</td>
<td>chtrace.h File Reference</td>
<td>681</td>
</tr>
<tr>
<td>9.64.1</td>
<td>Detailed Description</td>
<td>683</td>
</tr>
<tr>
<td>9.65</td>
<td>chversion.h File Reference</td>
<td>683</td>
</tr>
<tr>
<td>9.65.1</td>
<td>Detailed Description</td>
<td>683</td>
</tr>
<tr>
<td>9.66</td>
<td>chvt.c File Reference</td>
<td>683</td>
</tr>
<tr>
<td>9.66.1</td>
<td>Detailed Description</td>
<td>684</td>
</tr>
<tr>
<td>9.67</td>
<td>chvt.h File Reference</td>
<td>684</td>
</tr>
<tr>
<td>9.67.1</td>
<td>Detailed Description</td>
<td>685</td>
</tr>
</tbody>
</table>

Index | 687
Chapter 1

ChibiOS/RT

1.1 Copyright

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

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

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

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

1.2 Introduction

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

1.3 Related Documents

- ChibiOS/RT General Architecture
Chapter 2

Kernel Concepts

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

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

2.1 Naming Conventions

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

2.2 API Name Suffixes

The suffix can be one of the following:

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

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

In ChibiOS/RT there are three logical interrupt classes:

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

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

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

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

2.4 System States

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

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

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

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

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

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

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

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

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

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

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

- **Halted.** All interrupt sources are disabled and system stopped into an infinite loop. This state can be reached if the debug mode is activated and an error is detected or after explicitly invoking chSysHalt().
Note that the above states are just **Logical States** that may have no real associated machine state on some architectures. The following diagram shows the possible transitions between the states:

![System States Diagram]

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

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

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

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

2.7 Priority Levels

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

- **IDLEPRIO**, this is the lowest priority level and is reserved for the idle thread, no other threads should share this priority level. This is the lowest numerical value of the priorities space.

- **LOWPRIO**, the lowest priority level that can be assigned to an user thread.

- **NORMALPRIO**, this is the central priority level for user threads. It is advisable to assign priorities to threads as values relative to **NORMALPRIO**, as example **NORMALPRIO-1** or **NORMALPRIO+4**, this ensures the portability of code should the numerical range change in future implementations.

- **HIGHPRIO**, the highest priority level that can be assigned to an user thread.

- **ABSPRO**, absolute maximum software priority level, it can be higher than **HIGHPRIO** but the numerical values above **HIGHPRIO** up to **ABSPRO** (inclusive) are reserved. This is the highest numerical value of the priorities space.
2.8 Thread Working Area

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

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

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

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

Module Index

3.1 Modules

Here is a list of all modules:

- Release and Licensing .................................................. 21
- Release Information ..................................................... 22
- Customer Information ................................................... 24
- License Settings .......................................................... 26
- RT Kernel ................................................................. 28
- Version Numbers and Identification ................................. 29
- Configuration ............................................................ 31
  - Options ................................................................. 32
  - Checks ................................................................. 54
  - Restrictions .......................................................... 55
- System ................................................................. 56
  - Port Interface ....................................................... 57
  - OS Types and Structures .......................................... 58
  - OS Instances ........................................................ 68
  - Runtime Faults Collection Unit .................................... 71
  - Lists and Queues .................................................... 74
  - Scheduler ............................................................ 89
- Base Kernel Services .................................................... 110
  - System Management ................................................ 111
  - Memory Alignment .................................................. 136
  - Time and Intervals .................................................. 139
  - Virtual Timers ....................................................... 153
  - Threads ............................................................... 180
  - Time Measurement .................................................. 223
- Synchronization .......................................................... 227
  - Counting Semaphores ................................................ 228
  - Mutexes .............................................................. 246
  - Condition Variables ................................................ 261
  - Event Flags .......................................................... 273
  - Synchronous Messages .............................................. 300
- Dynamic Threads ........................................................ 311
- Registry ................................................................. 314
- Debug ................................................................. 324
- Checks and Assertions ................................................ 325
<table>
<thead>
<tr>
<th>Module</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tracing</td>
<td>335</td>
</tr>
<tr>
<td>Statistics</td>
<td>345</td>
</tr>
<tr>
<td>OS Library</td>
<td>349</td>
</tr>
<tr>
<td>Version Numbers and Identification</td>
<td>350</td>
</tr>
<tr>
<td>Synchronization</td>
<td>353</td>
</tr>
<tr>
<td>Binary Semaphores</td>
<td>354</td>
</tr>
<tr>
<td>Mailboxes</td>
<td>365</td>
</tr>
<tr>
<td>Pipes</td>
<td>383</td>
</tr>
<tr>
<td>Delegate Threads</td>
<td>391</td>
</tr>
<tr>
<td>Jobs Queues</td>
<td>403</td>
</tr>
<tr>
<td>Memory Management</td>
<td>418</td>
</tr>
<tr>
<td>Core Memory Manager</td>
<td>419</td>
</tr>
<tr>
<td>Memory Heaps</td>
<td>430</td>
</tr>
<tr>
<td>Memory Pools</td>
<td>437</td>
</tr>
<tr>
<td>Complex Services</td>
<td>462</td>
</tr>
<tr>
<td>Objects FIFOs</td>
<td>463</td>
</tr>
<tr>
<td>Objects Caches</td>
<td>480</td>
</tr>
<tr>
<td>Dynamic Objects Factory</td>
<td>490</td>
</tr>
</tbody>
</table>
Chapter 4

Hierarchical Index

4.1 Class Hierarchy

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

ch_delta_list ............................................................ 516
ch_dyn_element ............................................................ 518
ch_dyn_list ............................................................... 519
ch_dyn_mailbox ........................................................... 519
ch_dyn_object ............................................................. 521
ch_dyn_objects_filo ....................................................... 522
ch_dyn_pipe ............................................................... 523
chDynSemaphore .......................................................... 525
ch_job_descriptor ......................................................... 526
ch_jobs_queue ............................................................. 527
ch_list ................................................................. 529
ch_mutex ................................................................. 530
ch_objects_cache ....................................................... 532
ch_objects_factory ..................................................... 535
ch_objects_filo ............................................................ 538
ch_oc_hash_header ..................................................... 539
ch_oc_fru_header ........................................................ 541
ch_oc_object ............................................................. 543
ch_os_instance ........................................................... 546
ch_os_instance_config .................................................. 549
ch_priority_queue ......................................................... 550
ch_queue ................................................................. 552
ch_ready_list ............................................................ 553
ch_registered_static_object ........................................... 554
ch_registry ............................................................... 556
ch_rfcu ................................................................. 557
ch_semaphore ............................................................ 558
  ch_binary_semaphore ................................................ 515
ch_system ............................................................... 559
ch_system_debug ......................................................... 562
ch_thread ............................................................... 563
ch_threads_queue ....................................................... 571
ch_virtual_timer ......................................................... 572
ch_virtual_timers_list .................................................. 574
chdebug_t ............................................................... 576
<table>
<thead>
<tr>
<th>Symbol</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>condition_variable</td>
<td>580</td>
</tr>
<tr>
<td>event_listener</td>
<td>581</td>
</tr>
<tr>
<td>event_source</td>
<td>583</td>
</tr>
<tr>
<td>guarded_memory_pool_t</td>
<td>585</td>
</tr>
<tr>
<td>heap_header</td>
<td>586</td>
</tr>
<tr>
<td>kernel_stats_t</td>
<td>588</td>
</tr>
<tr>
<td>mailbox_t</td>
<td>590</td>
</tr>
<tr>
<td>memcore_t</td>
<td>593</td>
</tr>
<tr>
<td>memory_heap</td>
<td>594</td>
</tr>
<tr>
<td>memory_pool_t</td>
<td>596</td>
</tr>
<tr>
<td>pipe_t</td>
<td>597</td>
</tr>
<tr>
<td>pool_header</td>
<td>601</td>
</tr>
<tr>
<td>thread_descriptor_t</td>
<td>602</td>
</tr>
<tr>
<td>time_measurement_t</td>
<td>604</td>
</tr>
<tr>
<td>tm_calibration_t</td>
<td>606</td>
</tr>
<tr>
<td>trace_buffer_t</td>
<td>607</td>
</tr>
<tr>
<td>trace_event_t</td>
<td>610</td>
</tr>
</tbody>
</table>
Chapter 5

Data Structure Index

5.1 Data Structures

Here are the data structures with brief descriptions:

- ch_binary_semaphore
  Binary semaphore type .................................................. 515
- ch_delta_list
  Delta list element and header structure .............................. 516
- ch_dyn_element
  Type of a dynamic object list element .................................. 518
- ch_dyn_list
  Type of a dynamic object list .......................................... 519
- ch_dyn_mailbox
  Type of a dynamic buffer object ....................................... 519
- ch_dyn_object
  Type of a dynamic buffer object ....................................... 519
- ch_dyn_objects_fifo
  Type of a dynamic buffer object ....................................... 521
- ch_dyn_semaphore
  Type of a dynamic semaphore ........................................... 523
- ch_job_descriptor
  Type of a job descriptor ................................................ 526
- ch_jobs_queue
  Type of a jobs queue ................................................... 527
- ch_list
  Structure representing a generic single link list header and element 529
- ch_mutex
  Mutex structure .......................................................... 530
- ch_objects_cache
  Structure representing a cache object ................................ 532
- ch_objects_factory
  Type of the factory main object ...................................... 535
- ch_objects_fifo
  Type of an objects FIFO ................................................ 538
- ch_oc_hash_header
  Structure representing an hash table element ......................... 539
- ch_oc_lru_header
  Structure representing an hash table element ......................... 541
<table>
<thead>
<tr>
<th>Data Structure Index</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ch_oc_object</strong></td>
</tr>
<tr>
<td><strong>ch_os_instance</strong></td>
</tr>
<tr>
<td><strong>ch_os_instance_config</strong></td>
</tr>
<tr>
<td><strong>ch_priority_queue</strong></td>
</tr>
<tr>
<td><strong>ch_queue</strong></td>
</tr>
<tr>
<td><strong>ch_ready_list</strong></td>
</tr>
<tr>
<td><strong>ch_registered_static_object</strong></td>
</tr>
<tr>
<td><strong>ch_registry</strong></td>
</tr>
<tr>
<td><strong>ch_rfcu</strong></td>
</tr>
<tr>
<td><strong>ch_semaphore</strong></td>
</tr>
<tr>
<td><strong>ch_system</strong></td>
</tr>
<tr>
<td><strong>ch_system_debug</strong></td>
</tr>
<tr>
<td><strong>ch_thread</strong></td>
</tr>
<tr>
<td><strong>ch_threads_queue</strong></td>
</tr>
<tr>
<td><strong>ch_virtual_timer</strong></td>
</tr>
<tr>
<td><strong>ch_virtual_timers_list</strong></td>
</tr>
<tr>
<td><strong>chdebug_t</strong></td>
</tr>
<tr>
<td><strong>condition_variable</strong></td>
</tr>
<tr>
<td><strong>event_listener</strong></td>
</tr>
<tr>
<td><strong>event_source</strong></td>
</tr>
<tr>
<td><strong>guarded_memory_pool_t</strong></td>
</tr>
<tr>
<td><strong>heap_header</strong></td>
</tr>
<tr>
<td><strong>kernel_stats_t</strong></td>
</tr>
<tr>
<td><strong>mailbox_t</strong></td>
</tr>
<tr>
<td><strong>memcore_t</strong></td>
</tr>
<tr>
<td><strong>memory_heap</strong></td>
</tr>
<tr>
<td><strong>memory_pool_t</strong></td>
</tr>
<tr>
<td><strong>pipe_t</strong></td>
</tr>
<tr>
<td><strong>pool_header</strong></td>
</tr>
</tbody>
</table>
5.1 Data Structures

thread_descriptor_t
Type of a thread descriptor ................................................. 602

time_measurement_t
Type of a Time Measurement object ........................................... 604

tm_calibration_t
Type of a time measurement calibration data ................................. 606

trace_buffer_t
Trace buffer header .................................................................. 607

trace_event_t
Trace buffer record .................................................................. 610
Chapter 6

File Index

6.1 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 Name</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>chfactory.h</td>
<td>ChibiOS objects factory structures and macros</td>
<td>634</td>
</tr>
<tr>
<td>chinstances.c</td>
<td>OS instances code</td>
<td>637</td>
</tr>
<tr>
<td>chinstances.h</td>
<td>OS instances macros and structures</td>
<td>637</td>
</tr>
<tr>
<td>chjobs.h</td>
<td>Jobs Queues structures and macros</td>
<td>637</td>
</tr>
<tr>
<td>chlib.h</td>
<td>ChibiOS/LIB main include file</td>
<td>639</td>
</tr>
<tr>
<td>chlicense.h</td>
<td>License Module macros and structures</td>
<td>640</td>
</tr>
<tr>
<td>chlists.h</td>
<td>Lists and Queues header</td>
<td>641</td>
</tr>
<tr>
<td>chmboxes.c</td>
<td>Mailboxes code</td>
<td>643</td>
</tr>
<tr>
<td>chmboxes.h</td>
<td>Mailboxes macros and structures</td>
<td>643</td>
</tr>
<tr>
<td>chmemcore.c</td>
<td>Core memory manager code</td>
<td>645</td>
</tr>
<tr>
<td>chmemcore.h</td>
<td>Core memory manager macros and structures</td>
<td>645</td>
</tr>
<tr>
<td>chmemheaps.c</td>
<td>Memory heaps code</td>
<td>646</td>
</tr>
<tr>
<td>chmemheaps.h</td>
<td>Memory heaps macros and structures</td>
<td>646</td>
</tr>
<tr>
<td>chmempools.c</td>
<td>Memory Pools code</td>
<td>647</td>
</tr>
<tr>
<td>chmempools.h</td>
<td>Memory Pools macros and structures</td>
<td>647</td>
</tr>
<tr>
<td>chmsg.c</td>
<td>Messages code</td>
<td>650</td>
</tr>
<tr>
<td>chmsg.h</td>
<td>Messages macros and structures</td>
<td>650</td>
</tr>
<tr>
<td>chmtx.c</td>
<td>Mutexes code</td>
<td>651</td>
</tr>
<tr>
<td>chmtx.h</td>
<td>Mutexes macros and structures</td>
<td>651</td>
</tr>
<tr>
<td>chobjcaches.c</td>
<td>Objects Caches code</td>
<td>652</td>
</tr>
<tr>
<td>chobjcaches.h</td>
<td>Objects Caches macros and structures</td>
<td>653</td>
</tr>
<tr>
<td>chobjects.h</td>
<td>Operating System Objects macros and structures</td>
<td>654</td>
</tr>
<tr>
<td>chobjfifos.h</td>
<td>Objects FIFO structures and macros</td>
<td>655</td>
</tr>
<tr>
<td>chpipes.c</td>
<td>Pipes code</td>
<td>658</td>
</tr>
<tr>
<td>chpipes.h</td>
<td>Pipes macros and structures</td>
<td>659</td>
</tr>
<tr>
<td>chport.h</td>
<td>Port wrapper header</td>
<td>660</td>
</tr>
<tr>
<td>chregistry.c</td>
<td>Threads registry code</td>
<td>660</td>
</tr>
<tr>
<td>chregistry.h</td>
<td>Threads registry macros and structures</td>
<td>661</td>
</tr>
<tr>
<td>chrestrictions.h</td>
<td>Licensing restrictions header</td>
<td>662</td>
</tr>
<tr>
<td>File</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------------------------------------------------</td>
<td>------</td>
</tr>
<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 7

Module Documentation

7.1 Release and Licensing

7.1.1 Detailed Description

Modules

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

7.2.1 Detailed Description

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

Macros

- `#define __CHIBIOS__`
  ChibiOS product identification macro.
- `#define CH_VERSION_STABLE 1`
  Stable release flag.
- `#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 "21.6.0"`
  ChibiOS version string.
- `#define CH_VERSION_YEAR 21`
  ChibiOS version release year.
- `#define CH_VERSION_MONTH 6`
  ChibiOS version release month.
- `#define CH_VERSION_PATCH 0`
  ChibiOS version patch number.
- `#define CH_VERSION_NICKNAME "Atrani"`
  ChibiOS version nickname.

7.2.2 Macro Definition Documentation

7.2.2.1 __CHIBIOS__

`#define __CHIBIOS__`

ChibiOS product identification macro.

7.2.2.2 CH_VERSION_STABLE

`#define CH_VERSION_STABLE 1`

Stable release flag.
### 7.2.2.3 CH_VERSION

```c
#define CH_VERSION "21.6.0"
```

ChibiOS version string.

### 7.2.2.4 CH_VERSION_YEAR

```c
#define CH_VERSION_YEAR 21
```

ChibiOS version release year.

### 7.2.2.5 CH_VERSION_MONTH

```c
#define CH_VERSION_MONTH 6
```

ChibiOS version release month.

### 7.2.2.6 CH_VERSION_PATCH

```c
#define CH_VERSION_PATCH 0
```

ChibiOS version patch number.

### 7.2.2.7 CH_VERSION_NICKNAME

```c
#define CH_VERSION_NICKNAME "Atrani"
```

ChibiOS version nickname.

### 7.2.2.8 CH_VERSION_DATE

```c
#define CH_VERSION_DATE {{{CH_VERSION_YEAR + 2000} * 100} + CH_VERSION_MONTH}
```

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

7.3.1 Detailed Description

This module encapsulates 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 (yyyymm).
- `#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

7.3.2 Macro Definition Documentation
7.3 Customer Information

7.3.2.1 CH_CUSTOMER_ID_STRING

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

7.3.2.2 CH_CUSTOMER_ID_CODE

#define CH_CUSTOMER_ID_CODE "xxxx-yyyy"
Customer code.

7.3.2.3 CH_CUSTOMER_LICENSE_EOS_DATE

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

7.3.2.4 CH_CUSTOMER_LICENSE_VERSION_YEAR

#define CH_CUSTOMER_LICENSE_VERSION_YEAR 99
Licensed branch year.

7.3.2.5 CH_CUSTOMER_LICENSE_VERSION_MONTH

#define CH_CUSTOMER_LICENSE_VERSION_MONTH 12
Licensed branch month.

7.3.2.6 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.

7.3.2.7 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).
7.4 License Settings

7.4.1 Detailed Description

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

Macros

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

Allowed Features Levels

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

Deployment Options

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

Licensing Options

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

7.4.2 Macro Definition Documentation
7.4 License Settings

7.4.2.1 CH_LICENSE_TYPE_STRING

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

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

7.4.2.2 CH_LICENSE_ID_STRING

#define CH_LICENSE_ID_STRING "N/A"

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

7.4.2.3 CH_LICENSE_ID_CODE

#define CH_LICENSE_ID_CODE "N/A"

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

7.4.2.4 CH_LICENSE_MODIFIABLE_CODE

#define CH_LICENSE_MODIFIABLE_CODE TRUE

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

7.4.2.5 CH_LICENSE_FEATURES

#define CH_LICENSE_FEATURES CH_FEATURES_FULL

Code functionality restrictions.

7.4.2.6 CH_LICENSE_MAX_DEPLOY

#define CH_LICENSE_MAX_DEPLOY CH_DEPLOY_UNLIMITED

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

7.5.1 Detailed Description

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

Modules

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

7.6.1 Detailed Description

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

Kernel related info.

Macros

- #define __CHIBIOS_RT__
  ChibiOS/RT identification macro.
- #define CH_KERNEL_STABLE 0
  Stable release flag.

ChibiOS/RT version identification

- #define CH_KERNEL_VERSION "7.0.0"
  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 0
  Kernel version patch number.

Constants for configuration options

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

7.6.2 Macro Definition Documentation

7.6.2.1 __CHIBIOS_RT__

#define __CHIBIOS_RT__

ChibiOS/RT identification macro.
7.6.2.2  CH_KERNEL_STABLE

#define CH_KERNEL_STABLE 0
Stable release flag.

7.6.2.3  CH_KERNEL_VERSION

#define CH_KERNEL_VERSION "7.0.0"
Kernel version string.

7.6.2.4  CH_KERNEL_MAJOR

#define CH_KERNEL_MAJOR 7
Kernel version major number.

7.6.2.5  CH_KERNEL_MINOR

#define CH_KERNEL_MINOR 0
Kernel version minor number.

7.6.2.6  CH_KERNEL_PATCH

#define CH_KERNEL_PATCH 0
Kernel version patch number.

7.6.2.7  FALSE

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

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

7.6.2.8  TRUE

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

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

7.7.1 Detailed Description

Modules

- Options
- Checks
- Restrictions
7.8  Options

7.8.1  Detailed Description

Kernel related settings and hooks.

System 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.
- `#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)`
7.8 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.

7.8.2 Macro Definition Documentation

7.8.2.1 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.
7.8.2.2 CH_CFG_ST_RESOLUTION

#define CH_CFG_ST_RESOLUTION 32

System time counter resolution.

Note

Allowed values are 16, 32 or 64 bits.

7.8.2.3 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.

7.8.2.4 CH_CFG_INTERVALS_SIZE

#define CH_CFG_INTERVALS_SIZE 32

Time intervals data size.

Note

Allowed values are 16, 32 or 64 bits.

7.8.2.5 CH_CFG_TIME_TYPES_SIZE

#define CH_CFG_TIME_TYPES_SIZE 32

Time types data size.

Note

Allowed values are 16 or 32 bits.
7.8.2.6 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.

7.8.2.7 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.

7.8.2.8 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.

7.8.2.9 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.
7.8.2.10 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.

7.8.2.11 CH_CFG_USE_TIMESTAMP

#define CH_CFG_USE_TIMESTAMP TRUE

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

Note
The default is TRUE.

7.8.2.12 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.

7.8.2.13 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.
7.8.2.14 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.

7.8.2.15 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.

7.8.2.16 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.

7.8.2.17 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.
7.8.2.18 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.

7.8.2.19 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.

7.8.2.20 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.
7.8.2.21 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.

7.8.2.22 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.

7.8.2.23 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.
7.8.2.24 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.

7.8.2.25 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.

7.8.2.26 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.
7.8.2.27 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.

7.8.2.28 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_SEMAPHORES.
Mutexes are recommended.

7.8.2.29 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.
Module Documentation

7.8.2.30 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.

7.8.2.31 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.

7.8.2.32 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.

7.8.2.33 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.
7.8 Options

7.8.2.34  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.

7.8.2.35  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.

7.8.2.36  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.

7.8.2.37  CH_CFG_FACTORY_OBJECTS_REGISTRY

#define CH_CFG_FACTORY_OBJECTS_REGISTRY TRUE

Enables the registry of generic objects.

7.8.2.38  CH_CFG_FACTORY_GENERIC_BUFFERS

#define CH_CFG_FACTORY_GENERIC_BUFFERS TRUE

Enables factory for generic buffers.
7.8.2.39 CH_CFG_FACTORY_SEMAPHORES

#define CH_CFG_FACTORY_SEMAPHORES TRUE

Enables factory for semaphores.

7.8.2.40 CH_CFG_FACTORY_MAILBOXES

#define CH_CFG_FACTORY_MAILBOXES TRUE

Enables factory for mailboxes.

7.8.2.41 CH_CFG_FACTORY_OBJ_FIFOS

#define CH_CFG_FACTORY_OBJ_FIFOS TRUE

Enables factory for objects FIFOs.

7.8.2.42 CH_CFG_FACTORY_PIPES

#define CH_CFG_FACTORY_PIPES TRUE

Enables factory for Pipes.

7.8.2.43 CH_DBG_STATISTICS

#define CH_DBG_STATISTICS FALSE

Debug option, kernel statistics.

Note

The default is FALSE.
7.8.2.44 **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.

7.8.2.45 **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.

7.8.2.46 **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.

7.8.2.47 **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.
7.8.2.48  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_DISABLED.

7.8.2.49  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.

7.8.2.50  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.
7.8 Options

7.8.2.51 CH_DBG_THREADS_PROFILING

#define CH_DBG_THREADS_PROFILING FALSE

Debug option, threads profiling.
If enabled then a field is added to the thread_t structure that counts the system ticks occurred while executing the thread.

Note
The default is FALSE.
This debug option is not currently compatible with the tickless mode.

7.8.2.52 CH_CFG_SYSTEM_EXTRA_FIELDS

#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.

7.8.2.53 CH_CFG_SYSTEM_INIT_HOOK

#define CH_CFG_SYSTEM_INIT_HOOK( )

Value:
{
    /* Add system initialization code here.*/
}

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

7.8.2.54 CH_CFG_OS_INSTANCE_EXTRA_FIELDS

#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.

7.8.2.55 CH_CFG_OS_INSTANCE_INIT_HOOK

#define CH_CFG_OS_INSTANCE_INIT_HOOK( oip )

Value:
{
    /* Add OS instance initialization code here.*/
}

OS instance initialization hook.
Parameters

| in  | oip | pointer to the os_instance_t structure |

---

### 7.8.2.56 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.

### 7.8.2.57 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 |

---

### 7.8.2.58 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.
7.8 Options

Options 51

Parameters

| in  | tp  | pointer to the thread_t structure |

7.8.2.59 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 |

7.8.2.60 CH_CFG_IRQ_PROLOGUE_HOOK

#define CH_CFG_IRQ_PROLOGUE_HOOK( )

Value:

\[
/* IRQ prologue code here. */
\]

ISR enter hook.

7.8.2.61 CH_CFG_IRQ_EPILOGUE_HOOK

#define CH_CFG_IRQ_EPILOGUE_HOOK( )

Value:

\[
/* IRQ epilogue code here. */
\]

ISR exit hook.

ChibiOS/RT
### 7.8.2.62 CH_CFG_IDLE_ENTER_HOOK

```c
#define CH_CFG_IDLE_ENTER_HOOK()
```

**Value:**

```c
{ /* Idle-enter code here. */
}
```

Idle thread enter hook.

**Note**

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

### 7.8.2.63 CH_CFG_IDLE_LEAVE_HOOK

```c
#define CH_CFG_IDLE_LEAVE_HOOK()
```

**Value:**

```c
{ /* Idle-leave code here. */
}
```

Idle thread leave hook.

**Note**

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

### 7.8.2.64 CH_CFG_IDLE_LOOP_HOOK

```c
#define CH_CFG_IDLE_LOOP_HOOK()
```

**Value:**

```c
{ /* Idle loop code here. */
}
```

Idle Loop hook.

This hook is continuously invoked by the idle thread loop.
7.8 Options

7.8.2.65 CH_CFG_SYSTEM_TICK_HOOK

#define CH_CFG_SYSTEM_TICK_HOOK( )

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

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

7.8.2.66 CH_CFG_SYSTEM_HALT_HOOK

#define CH_CFG_SYSTEM_HALT_HOOK( reason )

Value:
{ /* System halt code here. */
}

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

7.8.2.67 CH_CFG_TRACE_HOOK

#define CH_CFG_TRACE_HOOK( step )

Value:
{ /* Trace code here. */
}

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

7.8.2.68 CH_CFG_RUNTIME_FAULTS_HOOK

#define CH_CFG_RUNTIME_FAULTS_HOOK( mask )

Value:
{ /* Faults handling code here. */
}

Runtime Faults Collection Unit hook.
This hook is invoked each time new faults are collected and stored.
7.9 Checks

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

7.10.1 Detailed Description

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

7.11.1 Detailed Description

Modules

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

7.12.1 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.
7.13 OS Types and Structures

7.13.1 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.

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

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
  Type of virtual timers list header.
• typedef struct ch_registry registry_t
7.13 OS Types and Structures

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.

7.13.2 Macro Definition Documentation

7.13.2.1 __CH_STRINGIFY

#define __CH_STRINGIFY(
    a)
"a"

Utility to make the parameter a quoted string.
7.13.2.2 __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.

7.13.2.3 __CH_USED

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

Marks an expression result as used.

Parameters

| in | x  | a valid expression |

7.13.2.4 likely

#define likely(x) PORT_LIKELY(x)

Marks a boolean expression as likely true.
7.13 OS Types and Structures

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

Parameters

\[
\text{in } \ x \quad \text{a valid expression}
\]

7.13.2.5 unlikely

\[
\text{#define unlikely(}
\quad \ x \quad \text{PORT_UNLIKELY}(x)
\]

Marks a boolean expression as likely false.

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

Parameters

\[
\text{in } \ x \quad \text{a valid expression}
\]

7.13.3 Typedef Documentation

7.13.3.1 rtcnt_t

typedef port_rtcnt_t rtcnt_t

Realtime counter.

7.13.3.2 rttime_t

typedef port_rttime_t rttime_t

Realtime accumulator.

7.13.3.3 syssts_t

typedef port_syssts_t syssts_t

System status word.
7.13.3.4 stkalign_t

typedef port_stkalign_t stkalign_t

Stack alignment type.

7.13.3.5 tmode_t

typedef uint8_t tmode_t

Thread flags.

7.13.3.6 tstate_t

typedef uint8_t tstate_t

Thread state.

7.13.3.7 trefs_t

typedef uint8_t trefs_t

Thread references counter.

7.13.3.8 tslices_t

typedef uint8_t tslices_t

Thread time slices counter.

7.13.3.9 tprio_t

typedef uint32_t tprio_t

Thread priority.

7.13.3.10 msg_t

typedef int32_t msg_t

Inter-thread message.
7.13.3.11 eventid_t

typedef int32_t eventid_t

Numeric event identifier.

7.13.3.12 eventmask_t

typedef uint32_t eventmask_t

Mask of event identifiers.

7.13.3.13 eventflags_t

typedef uint32_t eventflags_t

Mask of event flags.

7.13.3.14 cnt_t

typedef int32_t cnt_t

Generic signed counter.

7.13.3.15 ucnt_t

typedef uint32_t ucnt_t

Generic unsigned counter.

7.13.3.16 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.
7.13.3.17  thread_t

typedef struct ch_thread thread_t

Type of a thread structure.

7.13.3.18  os_instance_t

typedef struct ch_os_instance os_instance_t

Type of an OS instance structure.

7.13.3.19  virtual_timer_t

typedef struct ch_virtual_timer virtual_timer_t

Type of a Virtual Timer.

7.13.3.20  vtfunc_t

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

Type of a Virtual Timer callback function.

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>p 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. |

7.13.3.21  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.

### 7.13.3.22  registry_t

typedef struct ch_registry registry_t

Type of a registry structure.

### 7.13.3.23  thread_reference_t

typedef thread_t* thread_reference_t

Type of a thread reference.

### 7.13.3.24  threads_queue_t

typedef struct ch_threads_queue threads_queue_t

Type of a threads queue.

### 7.13.3.25  ready_list_t

typedef struct ch_ready_list ready_list_t

Type of a ready list header.

### 7.13.3.26  os_instance_config_t

typedef struct ch_os_instance_config os_instance_config_t

Type of a system instance configuration.
7.13.3.27  ch_system_t

typedef struct ch_system ch_system_t

Type of system data structure.

7.13.4  Enumeration Type Documentation

7.13.4.1  system_state_t

enum system_state_t

Global state of the operating system.

7.13.5  Function Documentation

7.13.5.1  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:

![Call Graph](chSysHalt __trace_halt)
7.14 OS Instances

7.14.1 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.

7.14.2 Macro Definition Documentation

7.14.2.1 __instance_get_currthread

`#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.

7.14.2.2 __instance_set_currthread

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

Current thread pointer set macro.
7.14.3 Function Documentation

7.14.3.1 __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

| in | p | the thread parameter, unused in this scenario |

7.14.3.2 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

| out | oip | pointer to the os_instance_t structure |
| in  | oicp | pointer to the os_instance_config_t structure |
Function Class:
Special function, this function has special requirements see the notes.

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

7.15.1 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.

7.15.2 Macro Definition Documentation

7.15.2.1 CH_RFCU_ALL_FAULTS

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

Mask of all faults.
7.15.3 Typedef Documentation

7.15.3.1 rfcu_mask_t

typedef uint32_t rfcu_mask_t

Type of a faults mask.

7.15.3.2 rfcu_t

typedef struct ch_rfcu rfcu_t

Type of an RFCU structure.

7.15.4 Function Documentation

7.15.4.1 chRFCUCollectFaultsI()

void chRFCUCollectFaultsI (  
    rfcu_mask_t mask )

Adds fault flags to the current mask.

Parameters

| in  | mask   | fault flags to be added |

7.15.4.2 chRFCUGetAndClearFaultsI()

rfcu_mask_t chRFCUGetAndClearFaultsI (  
    rfcu_mask_t mask )

Returns the current faults mask clearing it.

Parameters

| in  | mask   | mask of faults to be read and cleared |
Returns

The current faults mask.

Return values

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
</table>
| 0 | if no faults were collected since last call to this function.

7.15.4.3 __rfcu_object_init()

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

Runtime Faults Collection Unit initialization.

Note

Internal use only.

Parameters

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>rfcup</td>
</tr>
</tbody>
</table>

Function Class:

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

7.16.1 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 *qp)
  
  Queue initialization.

- static bool ch_queue_isempty (const ch_queue_t *qp)
7.16 Lists and Queues

• static bool ch_queue_notempty (const ch_queue_t *qp)
  Evaluates to true if the specified queue is 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.

7.16.2 Macro Definition Documentation

7.16.2.1 __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.
### 7.16.2.2 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

---

### 7.16.3 Typedef Documentation

#### 7.16.3.1 ch_list_t

```c
typedef struct ch_list ch_list_t
```

Type of a generic single link list header and element.

#### 7.16.3.2 ch_queue_t

```c
typedef struct ch_queue ch_queue_t
```

Type of a generic bidirectional linked list header and element.

#### 7.16.3.3 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.
7.16 Lists and Queues

7.16.3.4 ch_delta_list_t

typedef struct ch_delta_list ch_delta_list_t

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

7.16.4 Function Documentation

7.16.4.1 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.

7.16.4.2 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.
7.16.4.3 ch_list_notempty()

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

Evaluates to true if the specified list is not empty.

Parameters

<table>
<thead>
<tr>
<th></th>
<th>lp</th>
<th>pointer to the list header</th>
</tr>
</thead>
</table>

Returns

The status of the list.

Function Class:

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

7.16.4.4 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

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

Function Class:

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

7.16.4.5 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.
7.16 Lists and Queues

Parameters

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

Returns

The removed element pointer.

Function Class:

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

7.16.4.6 ch_queue_init()

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

Queue initialization.

Parameters

| out | qp | pointer to the queue header |

Function Class:

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

7.16.4.7 ch_queue_isempty()

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

Evaluates to true if the specified queue is empty.

Parameters

| 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.

7.16.4.8 ch_queue_notempty()

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

Evaluates to true if the specified queue is not empty.

Parameters
| 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.

7.16.4.9 ch_queue_insert()

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.

7.16.4.10 ch_queue_fifo_remove()

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

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

Parameters
\textbf{in} \textbf{qp} \textbf{the pointer to the queue list header}

Returns
The removed element pointer.

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

7.16.4.11 \textbf{ch\_queue\_lifo\_remove()}

\texttt{static ch\_queue\_t* ch\_queue\_lifo\_remove (}
\texttt{\textbf{ch\_queue\_t} * \textbf{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
\textbf{in} \textbf{qp} \textbf{the pointer to the queue list header}

Returns
The removed element pointer.

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

static ch_queue_t* ch_queue_dequeue (ch_queue_t* p) [inline], [static]

Removes an element from a queue and returns it.

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

Parameters

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

Returns

The removed element pointer.

Function Class:

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

7.16.4.13  ch_pqueue_init()

static void ch_pqueue_init (ch_priority_queue_t* pqp) [inline], [static]

Priority queue initialization.

Note

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

Parameters

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

Function Class:

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

7.16.4.14  ch_pqueue_remove_highest()

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.
7.16 Lists and Queues

Parameters

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

Returns

The removed element pointer.

Function Class:

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

7.16.4.15 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.

7.16.4.16 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

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

Returns

The inserted element pointer.

Function Class:

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

7.16.4.17  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.

7.16.4.18  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.
7.16 Lists and Queues

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

7.16.4.19 ch_dist_notempty()

static bool ch_dist_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.

7.16.4.20 ch_dist_islast()

static bool ch_dist_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.

7.16.4.21 ch_dist_isfirst()

static bool ch_dist_isfirst (  

ch_delta_list_t * dlhp,
ch_delta_list_t * dlp) [inline], [static]

Fist 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.

### 7.16.4.22 ch_dlist_insert_after()

```c
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

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

Function Class:

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

### 7.16.4.23 ch_dlist_insert_before()

```c
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

| in | dlhp | pointer to the delta list header element |
| in | dlp  | element to be inserted before the header element |
| in | delta | delta of the element to be inserted |
7.16 Lists and Queues

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

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

7.16.4.25  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

|       | dlhp         | pointer to the delta list header |

ChibiOS/RT
Function Class:

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

### 7.16.4.26 ch_dlist_dequeue()

```c
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.
7.17 Scheduler

7.17.1 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_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)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 *chSchSelectFirstI (void)`
  Makes runnable the first thread in the ready list, does not reschedule internally.
7.17.2  Macro Definition Documentation

7.17.2.1  MSG_OK

#define MSG_OK (msg_t)0

Normal wakeup message.

7.17.2.2  MSG_TIMEOUT

#define MSG_TIMEOUT (msg_t)-1

Wakeup caused by a timeout condition.

7.17.2.3  MSG_RESET

#define MSG_RESET (msg_t)-2

Wakeup caused by a reset condition.

7.17.2.4  NOPRIO

#define NOPRIO (tprio_t)0

Ready list header priority.

7.17.2.5  IDLEPRIO

#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.

#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.
### 7.17.2.12 CH_STATE_SUSPENDED

```c
#define CH_STATE_SUSPENDED (tstate_t)3
```

Suspended state.

### 7.17.2.13 CH_STATE_QUEUED

```c
#define CH_STATE_QUEUED (tstate_t)4
```

On a queue.

### 7.17.2.14 CH_STATE_WTSEM

```c
#define CH_STATE_WTSEM (tstate_t)5
```

On a semaphore.

### 7.17.2.15 CH_STATE_WTMTX

```c
#define CH_STATE_WTMTX (tstate_t)6
```

On a mutex.

### 7.17.2.16 CH_STATE_WTCOND

```c
#define CH_STATE_WTCOND (tstate_t)7
```

On a cond. variable.

### 7.17.2.17 CH_STATE_SLEEPING

```c
#define CH_STATE_SLEEPING (tstate_t)8
```

Sleeping.
7.17 Scheduler

7.17.2.18 CH_STATE_WTEXIT

#define CH_STATE_WTEXIT (tstate_t)9
Waiting a thread.

7.17.2.19 CH_STATE_WTOREVT

#define CH_STATE_WTOREVT (tstate_t)10
One event.

7.17.2.20 CH_STATE_WTANDEVT

#define CH_STATE_WTANDEVT (tstate_t)11
Several events.

7.17.2.21 CH_STATE_SNDMSGQ

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

7.17.2.22 CH_STATE_SNDMSG

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

7.17.2.23 CH_STATE_WTMSG

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

#define CH_STATE_FINAL (tstate_t)15

Thread terminated.

7.17.2.25 CH_STATE_NAMES

#define CH_STATE_NAMES

Value:
"READY", "CURRENT", "WTSTART", "SUSPENDED", "QUEUED", "WTSEM", "WIMTX", ..."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.

7.17.2.26 CH_FLAG_MODE_MASK

#define CH_FLAG_MODE_MASK (tmode_t)3U

Thread memory mode mask.

7.17.2.27 CH_FLAG_MODE_STATIC

#define CH_FLAG_MODE_STATIC (tmode_t)0U

Static thread.

7.17.2.28 CH_FLAG_MODE_HEAP

#define CH_FLAG_MODE_HEAP (tmode_t)1U

Thread allocated from a Memory Heap.
7.17.2.29  CH_FLAG_MODE_MPOOL

#define CH_FLAG_MODE_MPOOL (tmode_t)2U

Thread allocated from a Memory Pool.

7.17.2.30  CH_FLAG_TERMINATE

#define CH_FLAG_TERMINATE (tmode_t)4U

Termination requested flag.

7.17.2.31  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.

7.17.2.32  __sch_get_currthread

#define __sch_get_currthread( ) __instance_get_currthread(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.

7.17.3  Function Documentation
7.17.3.1 __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

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

Returns

The thread pointer.

Function Class:

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

Here is the call graph for this function:

```
   __sch_ready_behind
       __trace_ready
           ch_pqueue_insert_behind
```

### 7.17.3.2 \_\_sch\_ready\_ahead()

```
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.
**Parameters**

```plaintext
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](image)

### 7.17.3.3 __sch_reschedule_behind()

```c
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:

![Call Graph](image)
7.17.3.4  \texttt{__sch\_reschedule\_ahead()}

\begin{verbatim}
static void __sch_reschedule_ahead {  
    void  
}
\end{verbatim}

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.

\textbf{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:

\begin{center}
\includegraphics[width=\textwidth]{call_graph.png}
\end{center}

7.17.3.5  \texttt{ch\_sch\_prio\_insert()}

\begin{verbatim}
static void ch_sch_prio_insert (  
    ch_queue_t * qp,  
    ch_queue_t * tp  
)
\end{verbatim}

Inserts a thread into a priority ordered queue.

\textbf{Note}

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

\textbf{Parameters}

\begin{tabular}{|c|c|}
\hline
in & \texttt{qp} & the pointer to the threads list header \\
\hline
in & \texttt{tp} & the pointer to the thread to be inserted in the list \\
\hline
\end{tabular}
Function Class:
Not an API, this function is for internal use only.

7.17.3.6  chSchReadyI()

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.
Function Class:

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

Here is the call graph for this function:

![Call Graph](image)

### 7.17.3.7 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

<table>
<thead>
<tr>
<th>in</th>
<th>newstate</th>
<th>the new thread state</th>
</tr>
</thead>
</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]

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

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

**Returns**

The wakeup message.

**Return values**

- `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:

```
chSchGoSleepTimeoutS
chDbgCheckClassS
chVTDoSetI
chSysHalt
chDbgCheckClassI
chVTGetSystemTimeX
vt_enqueue
```

### 7.17.3.9 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>ntp</td>
</tr>
<tr>
<td>in</td>
<td>msg</td>
</tr>
</tbody>
</table>
Function Class:

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

Here is the call graph for this function:

```
7.17.3.10 chSchRescheduleS()

void chSchRescheduleS()
{
    void
}

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

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:
```
### 7.17.3.11 chSchIsPreemptionRequired()

```c
bool chSchIsPreemptionRequired (
    void )
```

Evaluates if preemption is required.

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

**Note**

Not a user function, it is meant to be invoked 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.

### 7.17.3.12 chSchDoPreemption()

```c
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:
7.17.3.13  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:

7.17.3.14  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:

![Call Graph](call_graph.png)

### 7.17.3.15 chSchSelectFirstI()

```c
thread_t * chSchSelectFirstI ( void )
```

Makes runnable the first 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:

![Call Graph](call_graph2.png)
7.18 Base Kernel Services

7.18.1 Detailed Description

Modules

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

7.19.1 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_IRQHandler(id) PORT_FAST_IRQHandler(id)
  
  Standard fast IRQ handler declaration.

Time conversion utilities for the realtime counter

- #define S2RTC(freq, sec) ((freq) \times (sec))
  Seconds to realtime counter.
- #define MS2RTC(freq, msec) (rtcnt_t)(((freq) + 999UL) / 1000UL) \times (msec)
  Milliseconds to realtime counter.
- #define US2RTC(freq, usec) (rtcnt_t)(((freq) + 999999UL) / 1000000UL) \times (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.
- bool chSysIntegrityCheckI (unsigned testmask)
  System integrity check.
- void chSysDisable (void)
  Raises the system interrupt priority mask to the maximum level.
- void chSysSuspend (void)
  Raises the system interrupt priority mask to system level.
• static void chSysEnable (void)
  Lower 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.
• static thread_t * chSysGetIdleThreadX (void)
  Returns a pointer to the idle thread.

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.

7.19.2 Macro Definition Documentation

7.19.2.1 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.
7.19.2.2 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.

7.19.2.3 currcore

#define currcore ch_system.instances[port_get_core_id()]

Access to current core's instance structure.

7.19.2.4 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
in prio the priority level

Returns
Priority range result.

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

7.19.2.5 CH_IRQ_IS_VALID_KERNEL_PRIORITY

#define CH_IRQ_IS_VALID_KERNEL_PRIORITY(
**7.19 System Management**

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. |

### 7.19.2.6 CH_IRQ_PROLOGUE

```c
#define CH_IRQ_PROLOGUE() 
        PORT_IRQ_PROLOGUE(); 
        CH_CFG_IRQ_PROLOGUE_HOOK(); 
        __stats_increase_irq(); 
        __trace_isr_enter(__func__); 
        __dbg_check_enter_isr() 
```

**Value:**

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

**Note**

Usually IRQ handlers functions are also declared naked.

On some architectures this macro can be empty.

**Function Class:**

- Special function, this function has special requirements see the notes.
7.19.2.7 CH_IRQ_EPILOGUE

#define CH_IRQ_EPILOGUE()

Value:
__dbg_check_leave_isr();
__trace_isr_leave(__func__);
CH_CFG_IRQ_EPILOGUE_HOOK();
PORT_IRQ_EPILOGUE()

IRQ handler exit code.

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

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

7.19.2.8 CH_IRQ_HANDLER

#define CH_IRQ_HANDLER(id)
PORT_IRQ_HANDLER(id)

Standard normal IRQ handler declaration.

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

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

7.19.2.9 CH_FAST_IRQ_HANDLER

#define CH_FAST_IRQ_HANDLER(id)
PORT_FAST_IRQ_HANDLER(id)

Standard fast IRQ handler declaration.

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

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

7.19.2.10 S2RTC

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

Seconds to realtime counter.

Converts from seconds to realtime counter cycles.

Note

The macro assumes that freq >= 1.

Parameters

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

Returns

The number of cycles.

Function Class:

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

7.19.2.11 MS2RTC

#define MS2RTC(
    freq,
    msec
) (rctnt_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 >= 1000.

Parameters

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

The number of cycles.

Function Class:

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

### 7.19.2.12 US2RTC

```c
#define US2RTC(freq, usec) ($(freq) + 999999UL) / 1000000UL \ast (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.

### 7.19.2.13 RTC2S

```c
#define RTC2S(freq, n) (((n) - 1UL) \div (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 \( freq \geq 1 \).
### 7.19 System Management

#### Parameters

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

**Returns**

The number of seconds.

**Function Class:**

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

#### 7.19.2.14 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 freq >= 1000.

**Parameters**

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

**Returns**

The number of milliseconds.

**Function Class:**

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

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

**Returns**

The number of microseconds.

**Function Class:**

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

7.19.2.16 chSysGetRealtimeCounterX

```c
#define chSysGetRealtimeCounterX() (rtcnt_t)port_rt_get_counter_value()
```

Returns the current value of the system real time counter.

**Note**

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

**Returns**

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

**Function Class:**

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

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

Value:

    __trace_switch(ntp, otp);
    __stats_ctxswc(ntp, otp);
    CH_CFG_CONTEXT_SWITCH_HOOK(ntp, otp);
    port_switch(ntp, otp);
```

Performs a context switch.

Note

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

Parameters

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

Function Class:

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

7.19.3 Function Documentation

7.19.3.1 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.

7.19.3.2 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.
7.19.3.3 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.*

7.19.3.4 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.*

7.19.3.5 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

\[
\text{in } \textit{reason} \quad \text{pointer to an error string}
\]

Function Class:

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

Here is the call graph for this function:

\[
\text{chSysHalt} \rightarrow \text{__trace_halt}
\]

7.19.3.6 \texttt{chSysIntegrityCheckI()}

\[
\text{bool chSysIntegrityCheckI (}
\quad \text{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 \texttt{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

\[
\text{in } \textit{testmask} \quad \text{Each bit in this mask is associated to a test to be performed.}
\]

Returns

The test result.
Return values

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

Function Class:

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

Here is the call graph for this function:

```
chSysIntegrityCheckI → chDbgCheckClassI → chSysHalt
```

### 7.19.3.7 chSysTimerHandlerI()

```c
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 `CH_CFG_TIME_QUANTUM` macro, the round robin interval.

Function Class:

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

7.19.3.8 chSysGetStatusAndLockX()

```c
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:

```
chSysGetStatusAndLockX
  \rightarrow chSysLock
  \rightarrow __dbg_check_lock

chSysLock
  \rightarrow __stats_start_measure
  \rightarrow _crit_thd

chSysLockFromISR
  \rightarrow __dbg_check_lock_from_isr
  \rightarrow __stats_start_measure
  \rightarrow _crit_isr
```

### 7.19.3.9 chSysRestoreStatusX()

```c
void chSysRestoreStatusX ( syssts_t sts )
```

Restores the specified execution status and leaves a critical zone.

**Note**

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

**Parameters**

| in  | sts | the system status to be restored. |
Function Class:

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

Here is the call graph for this function:

```
chSysRestoreStatusX
    \_sch\_reschedule\_ahead
    \_dbg\_check\_unlock
    \_stats\_stop\_measure
    _crit\_thd
    \_dbg\_check\_unlock
    \_from\_ISR
    \_stats\_stop\_measure
    _crit\_ISR
```

### 7.19.3.10 chSysIsCounterWithinX()

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

Realtime window test.

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

**Note**

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

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

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>cnt</th>
<th>the counter value to be tested</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></th>
<th>current time within the specified time window.</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td></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.

### 7.19.3.11 chSysPolledDelayX()

```c
void chSysPolledDelayX ( 
    rtcnt_t cycles 
)
```

Polled delay.

**Note**

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

**Parameters**

| in   | cycles | number of cycles |

Function Class:

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

Here is the call graph for this function:

![Call Graph]

### 7.19.3.12 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:

![Call Graph]

7.19.3.13 chSysSuspend()

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:

![Call Graph]
7.19.3.14 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:

```
chSysEnable —> __dbg_check_enable —> chSysHalt
```

7.19.3.15 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:

```
chSysLock —> __dbg_check_lock —> chSysHalt
```

7.19.3.16 chSysUnlock()

```c
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:

```
chSysUnlock
__dbg_check_unlock
__stats_stop_measure
_crit_thd
chSysHalt
chTMStopMeasurementX
```

7.19.3.17 chSysLockFromISR()

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

Enters the kernel lock state from within an interrupt handler.

**Note**

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

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

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:

```
<table>
<thead>
<tr>
<th>Function</th>
<th>Call Graph</th>
</tr>
</thead>
<tbody>
<tr>
<td>chSysLockFromISR</td>
<td>__dbg_check_lock_from_isr → chSysHalt</td>
</tr>
<tr>
<td>__dbg_check_lock_from_isr</td>
<td>__stats_start_measure → _crit_isr</td>
</tr>
<tr>
<td>__stats_start_measure</td>
<td>_crit_isr → chTMStartMeasurementX</td>
</tr>
</tbody>
</table>
```

### 7.19.3.18 chSysUnlockFromISR()

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

Leaves the kernel lock state from within an interrupt handler.

**Note**

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

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

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:

```
<table>
<thead>
<tr>
<th>Function</th>
<th>Call Graph</th>
</tr>
</thead>
<tbody>
<tr>
<td>chSysUnlockFromISR</td>
<td>__dbg_check_unlock_from_isr → chSysHalt</td>
</tr>
<tr>
<td>__dbg_check_unlock_from_isr</td>
<td>__stats_stop_measure → _crit_isr</td>
</tr>
<tr>
<td>__stats_stop_measure</td>
<td>_crit_isr → chTMStopMeasurementX</td>
</tr>
</tbody>
</table>
```
### 7.19.3.19 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 for chSysUnconditionalLock](image)

### 7.19.3.20 chSysUnconditionalUnlock()

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

Unconditionally leaves the kernel lock state.

**Note**

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

**Function Class:**

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

Here is the call graph for this function:

![Call Graph for chSysUnconditionalUnlock](image)
### 7.19.3.21 chSysNotifyInstance()

```c
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 |

### 7.19.3.22 chSysGetIdleThreadX()

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

Returns a pointer to the idle thread.

**Precondition**

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

**Note**

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

**Returns**

Pointer to the idle thread.

**Function Class:**

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

### 7.19.4 Variable Documentation

#### 7.19.4.1 ch_system

```c
ch_system_t ch_system
```

System root object.
7.19 System Management

7.19.4.2 ch0

CH_SYS_CORE0_MEMORY os_instance_t ch0

Core 0 OS instance.

7.19.4.3 ch_core0_cfg

const os_instance_config_t ch_core0_cfg

Initial value:

= {
  .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.

7.19.4.4 ch1

CH_SYS_CORE1_MEMORY os_instance_t ch1

Core 1 OS instance.

7.19.4.5 ch_core1_cfg

const os_instance_config_t ch_core1_cfg

Initial value:

= {
  .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.
7.20 Memory Alignment

7.20.1 Detailed Description

Memory Alignment services.

Memory alignment support macros

- \#define \texttt{MEM\_ALIGN\_MASK}(a) ((size\_t)(a) - 1U)
  
  Alignment mask constant.

- \#define \texttt{MEM\_ALIGN\_PREV}(p, a)
  
  Aligns to the previous aligned memory address.

- \#define \texttt{MEM\_ALIGN\_NEXT}(p, a)
  
  Aligns to the next aligned memory address.

- \#define \texttt{MEM\_IS\_ALIGNED}(p, a) (((size\_t)(p) & MEM\_ALIGN\_MASK(a)) == 0U)
  
  Returns whatever a pointer or memory size is aligned.

- \#define \texttt{MEM\_IS\_VALID\_ALIGNMENT}(a) (((size\_t)(a) !0U) & ((size\_t)(a) & (size\_t)(a) - 1U) == 0U)
  
  Returns whatever a constant is a valid alignment.

7.20.2 Macro Definition Documentation

7.20.2.1 MEM\_ALIGN\_MASK

\texttt{#define MEM\_ALIGN\_MASK(}

\texttt{a} \texttt{) ((size\_t)(a) - 1U)}

Alignment mask constant.

Parameters

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

7.20.2.2 MEM\_ALIGN\_PREV

\texttt{#define MEM\_ALIGN\_PREV(}

\texttt{p,}

\texttt{a} \texttt{)}

Value:

\texttt{/*lint -save -e9033 [10.8] The cast is safe.*/}

\texttt{(size\_t)(p) & ~MEM\_ALIGN\_MASK(a)}

\texttt{/*lint -restore*/}

Aligns to the previous aligned memory address.
7.20 Memory Alignment

Parameters

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

7.20.2.3 MEM_ALIGN_NEXT

#define MEM_ALIGN_NEXT(
  p, 
  a)

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

Aligns to the next aligned memory address.

Parameters

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

7.20.2.4 MEM_IS_ALIGNED

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

Returns whatever a pointer or memory size is aligned.

Parameters

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

7.20.2.5 MEM_IS_VALID_ALIGNMENT

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

Returns whatever a constant is a valid alignment.

Valid alignments are powers of two.
Parameters

| in | a  | alignment to be checked, must be a constant |
7.21 Time and Intervals

7.21.1 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) \times (time_conv_t)CH_CFG_ST_FREQUENCY))
  Seconds to time interval.
- \#define TIME_MS2I(msecs)
  Milliseconds to time interval.
- \#define TIME_US2I(usecs)
  Microseconds to time interval.
- \#define TIME_I2S(interval)
  Time interval to seconds.
- \#define TIME_I2MS(interval)
  Time interval to milliseconds.
- \#define TIME_I2US(interval)
  Time interval to microseconds.

Secure time conversion utilities

- static sysinterval_t chTimeS2I (time_secs_t secs)
  Seconds to time interval.
- static sysinterval_t chTimeMS2I (time_msecs_t msec)
  Milliseconds to time interval.
- static sysinterval_t chTimeUS2I (time_usecs_t usec)
  Microseconds to time interval.
- static time_secs_t chTimeI2S (sysinterval_t interval)
  Time interval to seconds.
- static time_msecs_t chTimeI2MS (sysinterval_t interval)
  Time interval to milliseconds.
- static time_usecs_t chTimeI2US (sysinterval_t interval)
  Time interval to microseconds.
- static systime_t chTimeAddX (systime_t systime, sysinterval_t interval)
  Adds an interval to a system time returning a system time.
• static sysinterval_t chTimeDiffX (systime_t start, systime_t end)  
  Subtracts two system times returning an interval.
• static bool chTimeIsInRangeX (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 chTimeStampsInRangeX (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.

7.21.2 Macro Definition Documentation

7.21.2.1 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.
7.21 Time and Intervals

7.21.2 TIME_INFINITE

#define TIME_INFINITE ((sysinterval_t)-1)

Infinite interval specification for all functions with a timeout specification.

Note

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

7.21.2.3 TIME_MAX_INTERVAL

#define TIME_MAX_INTERVAL ((sysinterval_t)-2)

Maximum interval constant usable as timeout.

7.21.2.4 TIME_MAX_SYSTIME

#define TIME_MAX_SYSTIME ((systime_t)-1)

Maximum system of system time before it wraps.

7.21.2.5 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

\[ \text{in} \quad \text{secs} \quad \text{number of seconds} \]
Returns

The number of ticks.

Function Class:

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

7.21.2.6 TIME_MS2I

#define TIME_MS2I(
    msecs )

Value:

{{sysinterval_t}(((time_conv_t)(msecs) * (time_conv_t)CH_CFG_ST_FREQUENCY) + (time_conv_t)999) / (time_conv_t)1000)}

Milliseconds to time interval.

Converts from milliseconds to system ticks number.

Note

The result is rounded upward to the next tick boundary.

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

Parameters

| 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.

7.21.2.7 TIME_US2I

#define TIME_US2I(
    usecs )

Value:

{{sysinterval_t}(((time_conv_t)(usecs) * (time_conv_t)CH_CFG_ST_FREQUENCY) + (time_conv_t)999999) / (time_conv_t)1000000)}

Microseconds to time interval.

Converts from microseconds to system ticks number.
7.21 Time and Intervals

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

\begin{verbatim}
in  usecs  number of microseconds
\end{verbatim}

Returns

The number of ticks.

Function Class:

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

7.21.2.8 TIME_I2S

\texttt{#define TIME_I2S(interval)}

\textbf{Value:}

\begin{verbatim}
(time_secs_t)((time_conv_t)(interval) + (time_conv_t)CH_CFG_ST_FREQUENCY - 1) / (time_conv_t)CH_CFG_ST_FREQUENCY
\end{verbatim}

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

\begin{verbatim}
in  interval  interval in ticks
\end{verbatim}

Returns

The number of seconds.

Function Class:

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

```c
#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**

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

**Returns**

The number of milliseconds.

**Function Class:**

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

### 7.21.2.10 TIME_I2US

```c
#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.
7.21 Time and Intervals

Parameters

| in | interval | interval in ticks |

Returns

The number of microseconds.

Function Class:

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

7.21.3 Typedef Documentation

7.21.3.1 systime_t

typedef uint64_t systime_t

Type of system time.

Note

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

7.21.3.2 sysinterval_t

typedef uint64_t sysinterval_t

Type of time interval.

Note

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

7.21.3.3 systimestamp_t

typedef uint64_t systimestamp_t

Type of a time stamp.
7.21.3.4  time_secs_t

typedef uint32_t time_secs_t
Type of seconds.
Note
   It is selectable in configuration between 16 or 32 bits.

7.21.3.5  time_msecs_t

typedef uint32_t time_msecs_t
Type of milliseconds.
Note
   It is selectable in configuration between 16 or 32 bits.

7.21.3.6  time_usecs_t

typedef uint32_t time_usecs_t
Type of microseconds.
Note
   It is selectable in configuration between 16 or 32 bits.

7.21.3.7  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.

7.21.4  Function Documentation

7.21.4.1  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.
7.21 Time and Intervals

Parameters

| in  | secs | number of seconds |

Returns

The number of ticks.

Function Class:

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

7.21.4.2 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.

7.21.4.3 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.
7.21.4.4 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.

7.21.4.5 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

| in | interval | interval in ticks |

Returns

The number of milliseconds.

Function Class:

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

### 7.21.4.6 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

| in | interval | interval in ticks |

Returns

The number of microseconds.

Function Class:

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

### 7.21.4.7 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.

7.21.4.8 chTimeDiffX()

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

Subtracts two system times returning an interval.

Parameters

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

Returns

The interval representing the time difference.

Function Class:

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

7.21.4.9 chTimeIsInRangeX()

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

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

Note

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

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

Return values

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

### 7.21.4.10 chTimeStampAddX()

```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><code>stamp</code></td>
<td>base time stamp</td>
</tr>
<tr>
<td><code>interval</code></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.

### 7.21.4.11 chTimeStampDiffX()

```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.
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.

### 7.21.4.12 chTimeStampIsInRangeX()

```c
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.
7.22 Virtual Timers

7.22.1 Detailed Description

Time and Virtual Timers related APIs and services.

Functions

- **static void vt_insert_first (virtual_timers_list_t vtp, 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 vtp, virtual_timer_t vtp, systime_t now, 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 chVTGetRemainingIntervalI (virtual_timer_t vtp)**

  Returns the remaining time interval before next timer trigger.

- **void chVTDoTickI (void)**

  Virtual timers ticker.

- **systimestamp_t chVTGetTimeStampl (void)**

  Generates a monotonic time stamp.

- **void chVTRestartTimeStampl (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 sysinterval_t chVTTimeElapsedSinceX (systime_t start)**

  Returns the elapsed time since the specified start time.

- **static bool chVTIsSystemTimeWithinX (systime_t start, systime_t end)**

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

- **static bool chVTIsSystemTimeWithin (systime_t start, systime_t end)**

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

- **static bool chVTGetTimersStateI (sysinterval_t timep)**

  Returns the time interval until the next timer event.

- **static bool chVTIsArmedI (const virtual_timer_t vtp)**

  Returns true if the specified timer is armed.

- **static bool chVTIsArmed (const virtual_timer_t vtp)**

  Returns true if the specified timer is armed.

- **static void chVTResetI (virtual_timer_t vtp)**

  Disables a Virtual Timer.

- **static void chVTReset (virtual_timer_t vtp)**

  Disables a Virtual Timer.

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

Resets and re-synchronizes the time stamps monotonic counter.

- static void __vt_object_init (virtual_timers_list_t *vtlp)

Virtual Timers instance initialization.

### 7.22.2 Function Documentation

#### 7.22.2.1 vt_insert_first()

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

Inserts a timer as first element in a delta list.

**Note**

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

Here is the call graph for this function:
7.22.2.2 vt_enqueue()

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

Enqueues a virtual timer in a virtual timers list.

Here is the call graph for this function:

![Call Graph](image)

7.22.2.3 chVTDoSetI()

```c
void chVTDoSetI (virtual_timer_t * vtp,
                 systime_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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>out</strong> vtp</td>
<td>the <code>virtual_timer_t</code> structure pointer</td>
</tr>
<tr>
<td><strong>in</strong> delay</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> is allowed but interpreted as a normal time specification.</td>
</tr>
<tr>
<td></td>
<td>• <code>TIME_IMMEDIATE</code> this value is not allowed.</td>
</tr>
<tr>
<td><strong>in</strong> vtfunc</td>
<td>the timer callback function. After invoking the callback the timer is disabled and the structure can be disposed or reused.</td>
</tr>
<tr>
<td><strong>in</strong> 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:

![Call Graph](image)

### 7.22.2.4 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.
7.22 Virtual Timers

Parameters

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

Function Class:

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

Here is the call graph for this function:

```
void chVTDoResetI ( virtual_timer_t * vtp )
```

Disables a Virtual Timer.

Precondition

The timer must be in armed state before calling this function.
7.22.2.6 chVTGetRemainingIntervalI()

```c
sysinterval_t chVTGetRemainingIntervalI (virtual_timer_t * vtp )
```

Returns the remaining time interval before next timer trigger.

**Note**

This function can be called while the timer is active.

**Parameters**

<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>
Returns

The remaining time interval.

Function Class:

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

Here is the call graph for this function:

![Call Graph](call_graph_image)

---

**7.22.2.7 chVTDoTickI()**

```c
void chVTDoTickI ( )
{
    void
}
```

Virtual timers ticker.

Note

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

Function Class:

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

### 7.22.2.8 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:

7.22.9 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:
7.22.10 chV TObjectInit()

static void chV TObjectInit ( virtual_timer_t * vtp ) [inline], [static]

Initializes a virtual_timer_t object.

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

Parameters

| Out  | vtp | the virtual_timer_t structure pointer |

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

7.22.11 chVTGetSystemTimeX()

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

Current system time.

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

Note
The counter can reach its maximum and then restart from zero.
This function can be called from any context but its atomicity is not guaranteed on architectures whose word size is less than systime_t size.

Returns
The system time in ticks.

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

7.22.2.12 chVTGetSystemTime()

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

Current system time.

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

**Note**

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

**Returns**

The system time in ticks.

**Function Class:**

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

Here is the call graph for this function:

![Call Graph Diagram]

7.22.2.13 chVTTimeElapsedSinceX()

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

Returns the elapsed time since the specified start time.

**Parameters**

- `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:

```
chVTTimeElapsedSinceX
  chTimeDiffX
  chVTGetSystemTimeX
```

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

**Return values**

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

Function Class:

This is an **X-Class** API, this function can be invoked from any context.
Here is the call graph for this function:

![Call Graph](image-url)

### 7.22.2.15 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**

| 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:

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

Here is the call graph for this function:

7.22.16  chVTGetTimersStateI()

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:

```
chVTGetTimersStateI
  chDbgCheckClassI
  chTimeDiffX
  chVTGetSystemTimeX
```

### 7.22.2.17 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**

```
| in vtp | the virtual_timer_t structure pointer |
```

Returns

true if the timer is armed.
Function Class:

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

Here is the call graph for this function:

```
chVTIsArmedI  chDbgCheckClassI  chSysHalt
```

### 7.22.2.18 chVTIsArmed()

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

Returns true if the specified timer is armed.

**Precondition**

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

**Parameters**

| in  | vtp | the virtual_timer_t structure pointer |

Returns

true if the timer is armed.
Function Class:

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

Here is the call graph for this function:

![Call Graph](image)

7.22.2.19 chVTResetI()

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

Disables a Virtual Timer.

Note

The timer is first checked and disabled only if armed.

Precondition

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

Parameters

| 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:

```
chVTReset
   ∼ chVTDoResetI
       ∼ chVTIsArmedI
           * ch_dlist_dequeue
             * ch_dlist_isempty
               * ch_dlist_isfirst
               * ch_dlist_remove_first
                   * chDbgCheckClassI
                   * chTimeAddX
                   * chTimeDiffX
                   * chVTGetSystemTimeX
                   * chVTIsArmed!
```

### 7.22.2.20 chVTReset()

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

Disables a Virtual Timer.

**Note**

The timer is first checked and disabled only if armed.

**Precondition**

The timer must have been initialized using `chV TObjectInit()` 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
__stats_stop_measure
__dbg_check_unlock
__stats_start_measure
__dbg_check_unlock
__stats_stop_measure
_chVTDoResetI
chVTIsArmedI
```

### 7.22.2.21 chVTSetI()

```c
static void chVTSetI (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 `chV TObjectInit()` 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>vtfunc</td>
<td>the timer callback function. After invoking the callback the timer is disabled and the structure can be disposed or reused.</td>
</tr>
<tr>
<td>in</td>
<td>par</td>
<td>a parameter that will be passed to the callback function</td>
</tr>
</tbody>
</table>
**Function Class:**

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

Here is the call graph for this function:

```
chVTSetI
chVTDoSetI
chVTResetI
chDbgCheckClassI
chVTGetSystemTimeX
vt_enqueue
chVTDoResetI
chVTIsArmedI
```

### 7.22.2.22 chVTSet()

```c
static void chVTSet (virtual_timer_t *vtp,
                     sysinterval_t delay,
                     vtfunc_t vtfun,
                     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>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>vtp</code></td>
<td>virtual_timer_t</td>
<td>the virtual_timer_t structure pointer</td>
</tr>
<tr>
<td><code>delay</code></td>
<td>sysinterval_t</td>
<td>the number of ticks before the operation timeouts, the special values are handled as follow:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- <code>TIME_INFINITE</code> is allowed but interpreted as a normal time specification.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- <code>TIME_IMMEDIATE</code> this value is not allowed.</td>
</tr>
<tr>
<td><code>vtfun</code></td>
<td>vtfunc_t</td>
<td>the timer callback function. After invoking the callback the timer is disabled and the structure can be disposed or reused.</td>
</tr>
<tr>
<td><code>par</code></td>
<td>void *</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:

```
chVTSet
chSysLock
__dbg_check_lock
__stats_start_measure
__dbg_check_unlock
__stats_stop_measure
chVTSetI
chVTDoSetI
chVTResetI
```

### 7.22.2.23 chVTSetContinuousI()

```c
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**

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

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>• <code>TIME_INFINITE</code> is allowed but interpreted as a normal time specification.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <code>TIME_IMMEDIATE</code> this value is not allowed.</td>
</tr>
<tr>
<td>in</td>
<td>vtfunc</td>
<td>the timer callback function. After invoking the callback the timer is disabled and the structure can be disposed or reused.</td>
</tr>
<tr>
<td>in</td>
<td>par</td>
<td>a parameter that will be passed to the callback function</td>
</tr>
</tbody>
</table>

**Function Class:**

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

Here is the call graph for this function:

```
7.22.2.24 chVTSetContinuous()

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 `chVObjectInit()` or `chVTDoSetI()`.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>vtp</th>
<th>the <code>virtual_timer_t</code> 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>• <code>TIME_INFINITE</code> is allowed but interpreted as a normal time specification.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <code>TIME_IMMEDIATE</code> this value is not allowed.</td>
</tr>
<tr>
<td>in</td>
<td>vtfunc</td>
<td>the timer callback function. After invoking the callback the timer is disabled and the structure can be disposed or reused.</td>
</tr>
</tbody>
</table>
Function Class:

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

Here is the call graph for this function:

7.22.2.25 chVTGetReloadIntervalX()

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

Returns the current reload value.

Parameters

| in | vtp | the virtual_timer_t structure pointer |

Returns

The reload value.

Function Class:

This is an **X-Class** API, this function can be invoked from any context.
7.22.2.26 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.

7.22.2.27 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:

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

7.22.2.28 chVTResetTimeStamp()

```c
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:

7.22.2.29 __vt_object_init()

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

Virtual Timers instance initialization.

Note

Internal use only.

Parameters

<table>
<thead>
<tr>
<th>out</th>
<th>vtlp</th>
<th>pointer to the virtual_timers_list_t structure</th>
</tr>
</thead>
</table>
Function Class:

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

Here is the call graph for this function:
7.23 Threads

7.23.1 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 *)((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.
7.23 Threads

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_memfill (uint8_t *startp, uint8_t *endp, uint8_t v)`
  Memory fill utility.
• `thread_t *chThdCreateSuspendedI (const thread_descriptor_t *tdp)`
  Creates a new thread into a static memory area.
• `thread_t *chThdCreateSuspended (const thread_descriptor_t *tdp)`
  Creates a new thread into a static memory area.
• `thread_t *chThdCreateI (const thread_descriptor_t *tdp)`
  Creates a new thread into a static memory area.
• `thread_t *chThdCreate (const thread_descriptor_t *tdp)`
  Creates a new thread into a static memory area.
• `thread_t *chThdCreateStatic (void *wsp, size_t size, tprio_t prio, tfunc_t pf, void *arg)`
  Creates a new thread into a static memory area.
- `thread_t * chThdStart (thread_t *tp)`
  Resumes a thread created with `chThdCreateI()`.
- `thread_t * chThdAddRef (thread_t *tp)`
  Adds a reference to a thread object.
- `void chThdRelease (thread_t *tp)`
  Releases a reference to a thread object.
- `void chThdExit (msg_t msg)`
  Terminates the current thread.
- `void chThdExitS (msg_t msg)`
  Terminates the current thread.
- `msg_t chThdWait (thread_t *tp)`
  Blocks the execution of the invoking thread until the specified thread terminates then the exit code is returned.
- `tprio_t chThdSetPriority (tprio_t newprio)`
  Changes the running thread priority level then reschedules if necessary.
- `void chThdTerminate (thread_t *tp)`
  Requests a thread termination.
- `void chThdSleep (sysinterval_t time)`
  Suspends the invoking thread for the specified time.
- `void chThdSleepUntil (systime_t time)`
  Suspends the invoking thread until the system time arrives to the specified value.
- `systime_t chThdSleepUntilWindowed (systime_t prev, systime_t next)`
  Suspends the invoking thread until the system time arrives to the specified value.
- `void chThdYield (void)`
  Yields the time slot.
- `msg_t chThdSuspendS (thread_reference_t *trp)`
  Sends the current thread sleeping and sets a reference variable.
- `msg_t chThdSuspendTimeoutS (thread_reference_t *trp, sysinterval_t timeout)`
  Sends the current thread sleeping and sets a reference variable.
- `void chThdResumeI (thread_reference_t *trp, msg_t msg)`
  Wakes up a thread waiting on a thread reference object.
- `void chThdResumeS (thread_reference_t *trp, msg_t msg)`
  Wakes up a thread waiting on a thread reference object.
- `void chThdResume (thread_reference_t *trp, msg_t msg)`
  Wakes up a thread waiting on a thread reference object.
- `msg_t chThdEnqueueTimeoutS (threads_queue_t *tqp, sysinterval_t timeout)`
  Enqueues the caller thread on a threads queue object.
- `void chThdDequeueNextI (threads_queue_t *tqp, msg_t msg)`
  Dequeues and wakes up one thread from the threads queue object, if any.
- `void chThdDequeueAllI (threads_queue_t *tqp, msg_t msg)`
  Dequeues and wakes up all threads from the threads queue object.
- `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 specified thread is in the CH_STATE_FINAL state.
7.23 Threads

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.

7.23.2 Macro Definition Documentation

7.23.2.1 __THREADS_QUEUE_DATA

#define __THREADS_QUEUE_DATA(name) {__CH_QUEUE_DATA(name)}

Data part of a static threads queue object initializer.

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

Parameters

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

7.23.2.2 THREADS_QUEUE_DECL

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

Static threads queue object initializer.

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

Parameters

| in | name | the name of the threads queue variable |
7.23.2.3 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.

7.23.2.4 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.

7.23.2.5 THD_WORKING_AREA_BASE

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

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

```c
#define THD_WORKING_AREA_END(
    s)

Value:

```

(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 |

### 7.23.2.7 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.

### 7.23.2.8 THD_DESCRIPTOR

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

Value:

```

{ (name),

  (wbase, wend, prio, funcp, arg) }
```

ChibiOS/RT
Thread descriptor initializer with no affinity.
Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>name</th>
<th>thread name</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>wbase</td>
<td>pointer to the working area base</td>
</tr>
<tr>
<td>in</td>
<td>wend</td>
<td>pointer to the working area end</td>
</tr>
<tr>
<td>in</td>
<td>prio</td>
<td>thread priority</td>
</tr>
<tr>
<td>in</td>
<td>funcp</td>
<td>thread function pointer</td>
</tr>
<tr>
<td>in</td>
<td>arg</td>
<td>thread argument</td>
</tr>
</tbody>
</table>

7.23.2.9 THD_DESCRIPTOR_AFFINITY

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

Value:
{
    (name),
    (wbase),
    (wend),
    (prio),
    (funcp),
    (arg),
    (oip)
}

Thread descriptor initializer with no affinity.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>name</th>
<th>thread name</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>wbase</td>
<td>pointer to the working area base</td>
</tr>
<tr>
<td>in</td>
<td>wend</td>
<td>pointer to the working area end</td>
</tr>
<tr>
<td>in</td>
<td>prio</td>
<td>thread priority</td>
</tr>
<tr>
<td>in</td>
<td>funcp</td>
<td>thread function pointer</td>
</tr>
<tr>
<td>in</td>
<td>arg</td>
<td>thread argument</td>
</tr>
<tr>
<td>in</td>
<td>oip</td>
<td>instance affinity</td>
</tr>
</tbody>
</table>

7.23.2.10 chThdSleepSeconds

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

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

Parameters

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

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

7.23.2.11 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.

7.23.2.12 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

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

in usc time in microseconds, must be different from zero

Function Class:

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

7.23.3 Typedef Documentation

7.23.3.1 tfunc_t

typedef void(* tfunc_t) (void *p)

Thread function.

7.23.4 Function Documentation

7.23.4.1 __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

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>oip</td>
<td></td>
</tr>
<tr>
<td>in</td>
<td>tp</td>
<td></td>
</tr>
<tr>
<td>in</td>
<td>name</td>
<td></td>
</tr>
<tr>
<td>in</td>
<td>prio</td>
<td></td>
</tr>
</tbody>
</table>

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:

![Call Graph]

7.23.4.2 __thd_memfill()

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

Memory fill utility.

Parameters

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

Function Class:

Not an API, this function is for internal use only.
7.23.4.3 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:

```
<table>
<thead>
<tr>
<th>chThdCreateSuspendedI</th>
<th>chDbgCheckClassI</th>
<th>ch_list_init</th>
</tr>
</thead>
<tbody>
<tr>
<td>_thd_object_init</td>
<td>ch_queue_init</td>
<td></td>
</tr>
<tr>
<td></td>
<td>chTMObjectInit</td>
<td></td>
</tr>
<tr>
<td></td>
<td>chSysHalt</td>
<td></td>
</tr>
</tbody>
</table>
```
7.23.4.4 \texttt{chThdCreateSuspended()}

\begin{verbatim}
thread_t * chThdCreateSuspended(
    const thread_descriptor_t * tdp )
\end{verbatim}

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 \texttt{CH\_STATE\_WTSTART}.

Postcondition

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

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

Note

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

Parameters

\begin{center}
\begin{tabular}{ll}
\textbf{out} & \texttt{tdp} & pointer to the thread descriptor
\end{tabular}
\end{center}

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:
7.23.4.5  chThdCreateI()

 thread_t * chThdCreateI (  
     const thread_descriptor_t * tdp )

Creates a new thread into a static memory area.

The new thread is initialized and make ready to execute.

Postcondition

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

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

Note

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

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

Parameters

out  tdp  pointer to the thread descriptor

Returns

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

Function Class:

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

Here is the call graph for this function:
7.23.4.6 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**

| 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.
7.23 Threads

Here is the call graph for this function:

### 7.23.4.7 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:

**7.23.4.8 chThdStart()**

```c
thread_t * chThdStart ( 
    thread_t * tp )
```

Resumes a thread created with `chThdCreateI()`.
### 7.23 Threads

**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:**

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

Here is the call graph for this function:

```
chThdStart
  \_sch\_ready\_ahead
  \_sch\_ready\_behind
  chSchWakeupS
  chDbgCheckClassS
  chSysNotifyInstance
  \_dbg\_check\_lock
  \_stats\_start\_measure
  \_crit\_thd
  \_dbg\_check\_unlock
  \_stats\_stop\_measure
  \_crit\_thd
  chSysLock
  chSysUnlock
```

### 7.23.4.9 chThdAddRef()

```c
thread_t * chThdAddRef ( thread_t * tp )
```

Adds a reference to a thread object.

**Precondition**

The configuration option CH_CFG_USE_REGISTRY must be enabled in order to use this function.
Parameters

| 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:

```
chThdAddRef
  \|-- chSysLock
     \|-- \_dbg_check_lock
      \|-- \_stats_start_measure
           \|-- _crit_thd

chThdAddRef
  \|-- chSysUnlock
     \|-- \_dbg_check_unlock

chThdRelease()

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

|   | 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:

```
chThdRelease
   ↓
   chPoolFree
   ↓
   chPoolFreeI
   ↓
   chSysUnlock
   ↓
   chThdGetWorkingAreaX
   ↓
   chHeapFree
   ↓
   chSysLock
```

7.23.4.11 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

|   | 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:

7.23.4.12 chThdExitS()

```c
void chThdExitS (msg_t msg)
```

Terminates the current thread.

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

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

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

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:

7.23.4.14 chThdSetPriority()

t prio_t chThdSetPriority ( 
    t prio_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.
### 7.23 Threads

#### Parameters

| in | newprio | the new priority level of the running thread |

#### Returns

The old priority level.

#### Function Class:

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

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

![Call Graph](image)

---

#### 7.23.4.15 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:

7.23.4.16 chThdSleep()

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

Suspends the invoking thread for the specified time.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>time</th>
<th>the delay in system ticks, the special values are handled as follow:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>• TIME_INFINITE the thread enters an infinite sleep state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• TIME_IMMEDIATE this value is not allowed.</td>
</tr>
</tbody>
</table>
Function Class:
Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

7.23.4.17 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

<table>
<thead>
<tr>
<th>in</th>
<th>time</th>
<th>absolute system time</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:

![Call Graph](image)

### 7.23.4.18 `chThdSleepUntilWindowed()`

```c
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>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>prev</code></td>
<td>absolute system time of the previous deadline</td>
</tr>
<tr>
<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:

7.23.4.19 chThdYield()

```c
void chThdYield ( 
    void )
```

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:

7.23.4.20 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

| in | trp | a pointer to a thread reference object |

Returns

The wake up message.
Function Class:

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

Here is the call graph for this function:

```
chThdSuspendS
  chSchGoSleepS
ch_pqueue_remove_highest
chDbgCheckClassS
  chSysGetIdleThreadX
  chThdGetSelfX
```

### 7.23.4.21 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>trp</th>
<th>a pointer to a thread reference 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>• <strong>TIME_INFINITE</strong> the thread enters an infinite sleep state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <strong>TIME_IMMEDIATE</strong> the thread is not suspended and the function returns</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>MSG_TIMEOUT</strong> as if a timeout occurred.</td>
</tr>
</tbody>
</table>

**Returns**

The wake up message.

**Return values**

- **MSG_TIMEOUT** if the operation timed out.
Function Class:

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

Here is the call graph for this function:

![Call graph](image)

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

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

Here is the call graph for this function:

![Call Graph Diagram]

```plaintext
chThdResumeI
chSchReadyI
__sch_ready_behind
chDbgCheckClassI
chSysNotifyInstance
```

### 7.23.4.23 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></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>in</strong></td>
<td><strong>trp</strong></td>
</tr>
<tr>
<td><strong>in</strong></td>
<td><strong>msg</strong></td>
</tr>
</tbody>
</table>
Function Class:

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

Here is the call graph for this function:

![Call Graph](call_graph.png)

### 7.23.4.24  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>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:

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

Here is the call graph for this function:

```
chThdResume
chSysLock
chSysUnlock
chThdResumeS
__dbg_check_lock
__stats_start_measure
_crit_thd
__dbg_check_unlock
__stats_stop_measure
_crit_thd
chSchWakeupS
```

### 7.23.4.25 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>• <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 enqueued and the function returns <code>MSG_TIMEOUT</code> as if a timeout occurred.</td>
</tr>
</tbody>
</table>

**Returns**

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

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

Function Class:

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

Here is the call graph for this function:

![Call Graph](image)

### 7.23.4.26 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**

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

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

Here is the call graph for this function:

```
chThdDequeueNextI
   ch_queue_notempty
   chThdDoDequeueNextI
   ch_queue_fifo_remove
   chSchReadyI
```

### 7.23.4.27 chThdDequeueAllI()

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

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

**Parameters**

<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:

```
chThdDequeueAllI
   ch_queue_notempty
   chThdDoDequeueNextI
   ch_queue_fifo_remove
   chSchReadyI
```
7.23.4.28  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.

7.23.4.29  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:

7.23.4.30  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.
7.23 Threads

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.

7.23.4.31 chThdGetWorkingAreaX()

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.

7.23.4.32 chThdTerminatedX()

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 |
Return values

<table>
<thead>
<tr>
<th>true</th>
<th>thread terminated.</th>
</tr>
</thead>
<tbody>
<tr>
<td>false</td>
<td>thread not terminated.</td>
</tr>
</tbody>
</table>

Function Class:

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

### 7.23.4.33 chThdShouldTerminateX()

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

Verifies if the current thread has a termination request pending.

Return values

<table>
<thead>
<tr>
<th>true</th>
<th>termination request pending.</th>
</tr>
</thead>
<tbody>
<tr>
<td>false</td>
<td>termination request not pending.</td>
</tr>
</tbody>
</table>

Function Class:

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

Here is the call graph for this function:

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

### 7.23.4.34 chThdStartI()

```c
static thread_t* chThdStartI (  
    thread_t * tp ) [inline], [static]
```

Resumes a thread created with `chThdCreateI()`.
7.23 Threads

Parameters

| in  | tp  | pointer to the thread |

Returns

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

Function Class:

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
```

7.23.4.35 chThdSleepS()

static void chThdSleepS (sysinterval_t ticks) [inline], [static]

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:

```
chThdSleepS ➔ chSchGoSleepTimeoutS ➔ chDbgCheckClassS ➔ chVTDoSetI
```

### 7.23.4.36 chThdQueueObjectInit()

```c
static void chThdQueueObjectInit (threads_queue_t *tqp) [inline], [static]
```

Initializes a threads queue object.

**Parameters**

- `tqp` pointer to the threads queue object

**Function Class:**

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

Here is the call graph for this function:

```
chThdQueueObjectInit ➔ ch_queue_init
```

### 7.23.4.37 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

\begin{tabular}{|l|c|}
\hline
false & if the queue is not empty. \\
true & if the queue is empty. \\
\hline
\end{tabular}

Function Class:

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

Here is the call graph for this function:

\begin{center}
\includegraphics[width=\textwidth]{call_graph.png}
\end{center}

### 7.23.4.38  chThdDoDequeueNextI()

```
static void chThdDoDequeueNextI (  
    threads_queue_t * tqp,  
    msg_t msg ) [inline], [static]
```

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

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

Precondition

The queue must contain at least an object.
Parameters

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

Function Class:

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

Here is the call graph for this function:
7.24 Time Measurement

7.24.1 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.

7.24.2 Macro Definition Documentation

7.24.2.1 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.

7.24.3 Function Documentation

7.24.3.1 chTMObjectInit()

`void chTMObjectInit (time_measurement_t *tmp)`

Initializes a TimeMeasurement object.
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 <code>TimeMeasurement</code> structure</td>
</tr>
</tbody>
</table>

Function Class:

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

7.24.3.2 chTMStartMeasurementX()

NOINLINE void chTMStartMeasurementX (    
    `time_measurement_t` * tmp )

Starts a measurement.

Precondition

The `time_measurement_t` structure must be initialized.

Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in,out</td>
<td>tmp</td>
<td>pointer to a <code>TimeMeasurement</code> structure</td>
</tr>
</tbody>
</table>

Function Class:

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

7.24.3.3 chTMStopMeasurementX()

NOINLINE void chTMStopMeasurementX (    
    `time_measurement_t` * tmp )

Stops a measurement.

Precondition

The `time_measurement_t` structure must be initialized.

Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in,out</td>
<td>tmp</td>
<td>pointer to a <code>time_measurement_t</code> structure</td>
</tr>
</tbody>
</table>
7.24 Time Measurement

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

### 7.24.3.4 chTMChainMeasurementToX()

```c
NOINLINE void chTMChainMeasurementToX (
    time_measurement_t ∗ tmp1,
    time_measurement_t ∗ tmp2 )
```

Stops a measurement and chains to the next one using the same time stamp.

**Parameters**

<table>
<thead>
<tr>
<th>Mode</th>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in,out</td>
<td><code>tmp1</code></td>
<td>pointer to the <code>time_measurement_t</code> structure to be stopped</td>
</tr>
<tr>
<td>in,out</td>
<td><code>tmp2</code></td>
<td>pointer to the <code>time_measurement_t</code> structure to be started</td>
</tr>
</tbody>
</table>

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

### 7.24.3.5 __tm_calibration_object_init()

```c
static void __tm_calibration_object_init ( tm_calibration_t ∗ tcp ) [inline], [static]
```

Time measurement initialization.

**Note**

Internal use only.

**Parameters**

<table>
<thead>
<tr>
<th>Mode</th>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td><code>tcp</code></td>
<td>pointer to the <code>tm_calibration_t</code> 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:
7.25 Synchronization

7.25.1 Detailed Description

Synchronization services.

Modules

- Counting Semaphores
- Mutexes
- Condition Variables
- Event Flags
- Synchronous Messages
7.26 Counting Semaphores

7.26.1 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.
7.26 Counting Semaphores

Data Structures

- struct ch_semaphore
  Semaphore structure.

Functions

- void chSemObjectInit (semaphore_t *sp, cnt_t n)
  Initializes a semaphore with the specified counter value.

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

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

- msg_t chSemWait (semaphore_t *sp)
  Performs a wait operation on a semaphore.

- msg_t chSemWaitS (semaphore_t *sp)
  Performs a wait operation on a semaphore.

- msg_t chSemWaitTimeout (semaphore_t *sp, sysinterval_t timeout)
  Performs a wait operation on a semaphore with timeout specification.

- msg_t chSemWaitTimeoutS (semaphore_t *sp, sysinterval_t timeout)
  Performs a wait operation on a semaphore with timeout specification.

- void chSemSignal (semaphore_t *sp)
  Performs a signal operation on a semaphore.

- void chSemSignalI (semaphore_t *sp)
  Performs a signal operation on a semaphore.

- void chSemAddCounterI (semaphore_t *sp, cnt_t n)
  Adds the specified value to the semaphore counter.

- msg_t chSemSignalWait (semaphore_t *sp, semaphore_t *spw)
  Performs atomic signal and wait operations on two semaphores.

- static void chSemReset (semaphore_t *sp, cnt_t n)
  Performs a reset operation on the semaphore.

- static void chSemResetI (semaphore_t *sp, cnt_t n)
  Performs a reset operation on the semaphore.

- static void chSemFastWaitI (semaphore_t *sp)
  Decreases the semaphore counter.

- static void chSemFastSignalI (semaphore_t *sp)
  Increases the semaphore counter.

- static cnt_t chSemGetCounterI (const semaphore_t *sp)
  Returns the semaphore counter current value.

7.26.2 Macro Definition Documentation

7.26.2.1 __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.
Parameters

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

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

7.26.3 Typedef Documentation

7.26.3.1 semaphore_t

typedef struct ch_semaphore semaphore_t

Semaphore structure.

7.26.4 Function Documentation

7.26.4.1 chSemObjectInit()

```c
void chSemObjectInit ( 
    semaphore_t * sp, 
    cnt_t n )
```

Initializes a semaphore with the specified counter value.
Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>out</code></td>
<td><code>sp</code></td>
</tr>
<tr>
<td><code>in</code></td>
<td><code>n</code></td>
</tr>
</tbody>
</table>

Function Class:

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

Here is the call graph for this function:

![Call Graph]

---

### 7.26.4.2 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>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>in</code></td>
<td><code>sp</code></td>
</tr>
<tr>
<td><code>in</code></td>
<td><code>n</code></td>
</tr>
<tr>
<td><code>in</code></td>
<td><code>msg</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:

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

Performs a reset operation on the semaphore.

Postcondition

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

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

Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>pointer to a semaphore_t structure</td>
</tr>
<tr>
<td>in</td>
<td>the new value of the semaphore counter. The value must be non-negative.</td>
</tr>
<tr>
<td>in</td>
<td>message to be sent</td>
</tr>
</tbody>
</table>
7.26 Counting Semaphores

Function Class:

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

Here is the call graph for this function:

```
chSemResetWithMessageI
  |     |
  v     v
ch_queue_isempty
  |     |
  v     v
ch_queue_lifo_remove
  |     |
  v     v
ch_queue_notempty
  |     |
  v     v
chDbgCheckClassI
  |     |
  v     v
chSchReadyI
  |     |
  v     v
chSysHalt
  |     |
  v     v
__sch_ready_behind
  |     |
  v     v
chSysNotifyInstance
```

7.26.4.4 **chSemWait()**

```c
msg_t chSemWait ( 
    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**

| 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()`.

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:

![Call Graph](image)

7.26.4.5 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>
<tbody>
<tr>
<td>MSG_RESET</td>
<td>if the semaphore has been reset using <code>chSemReset()</code> .</td>
</tr>
</tbody>
</table>

Function Class:

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

Here is the call graph for this function:

```
chSemWaitS
  \-- ch_queue_isempty
  \-- ch_queue_notempty
    \-- chDbgCheckClassS
    \-- chThdGetSelfX
```

### 7.26.4.6 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>• <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

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:

![Call Graph](image)

### 7.26.4.7 chSemWaitTimeoutS()

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

Performs a wait operation on a semaphore with timeout specification.
Parameters

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

Returns

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

Return values

<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:

7.26.4.8 chSemSignal()

void chSemSignal ( semaphore_t * sp )

Performs a signal operation on a semaphore.
Parameters

| in | sp | pointer to a semaphore_t structure |

Function Class:

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

Here is the call graph for this function:

7.26.4.9 chSemSignalI()

void chSemSignalI (  
    semaphore_t * sp  )

Performs a signal operation on a semaphore.

Postcondition

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

| in  | sp | pointer to a semaphore_t structure |

Function 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:

```
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:

```
chSemAddCounterI
  |   |   |
  v   v   v
ch_queue_fifo_remove
  |   |   |
  v   v   v
ch_queue_isempty
  |   |   |
  v   v   v
ch_queue_notempty
  |   |   |
  v   v   v
chDbgCheckClassI
  |   |   |
  v   v   v
chSchReadyI
  |   |   |
  v   v   v
chSysHalt
  |   |   |
  v   v   v
__sch_ready_behind
  |   |   |
  v   v   v
chSysNotifyInstance
```

### 7.26.4.11 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**

- **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()`.
7.26 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:

```
7.26.4.12 chSemReset()

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

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

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

Here is the call graph for this function:

```
chSemReset
    ↓
chSemResetWithMessage
    ↓
chSchRescheduleS
    ↓
chSemResetWithMessageI
    ↓
chSysLock
    ↓
chSysUnlock
```

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

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

Here is the call graph for this function:

![Call Graph for chSemFastWaitI](image)

### 7.26.4.14 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**

| in | sp | pointer to a semaphore_t structure |

**Function Class:**

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

Here is the call graph for this function:

![Call Graph for chSemFastWaitI](image)
7.26.4.15  chSemFastSignalI()

static void chSemFastSignalI (  
    semaphore_t * sp ) [inline], [static]  

Increases the semaphore counter.

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

Parameters

in  sp  pointer to a semaphore_t structure

Function Class:

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

Here is the call graph for this function:

7.26.4.16  chSemGetCounterI()

static cnt_t chSemGetCounterI (  
    const semaphore_t * sp ) [inline], [static]  

Returns the semaphore counter current value.

Parameters

in  sp  pointer to a semaphore_t structure

Returns

The semaphore counter value.
Function Class:

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

Here is the call graph for this function:
7.27 Mutexes

7.27.1 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.

7.27.2 Macro Definition Documentation
7.27.2.1 __MUTEX_DATA

#define __MUTEX_DATA(
    name ) {__CH_QUEUE_DATA(name.queue), NULL, NULL, 0}

Data part of a static mutex initializer.

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

| in  | name | the name of the mutex variable |

### 7.27.2.2 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 |

### 7.27.3 Typedef Documentation

#### 7.27.3.1 mutex_t

```c
typedef struct ch_mutex mutex_t
```

Type of a mutex structure.

### 7.27.4 Function Documentation

#### 7.27.4.1 chMtxObjectInit()

```c
void chMtxObjectInit ( mutex_t * mp )
```

Initializes a `mutex_t` structure.

Parameters

| out | mp | pointer to a `mutex_t` structure |
Function Class:

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

Here is the call graph for this function:

```
chMtxObjectInit  ch_queue_init
```

### 7.27.4.2 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.
7.27 Mutexes

Here is the call graph for this function:

```
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](image)

### 7.27.4.4 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:

```
chMtxTryLockS

chMtxTryLock

chSysLock

chSysUnlock

chDbgCheckClassS

chThdGetSetIX

__dbg_check_lock

__stats_start_measure

_crit_thd

__dbg_check_unlock

__stats_stop_measure

_crit_thd
```

7.27.4.5 chMtxTryLockS()

```c
bool chMtxTryLockS (    

  mutex_t * mp   )
```

Tries to lock a mutex.

This function attempts to lock a mutex, if the mutex is already taken by another thread then the function exits without waiting.

Postcondition

The mutex is locked and inserted in the per-thread stack of owned mutexes.

Note

This function does not have any overhead related to the priority inheritance mechanism because it does not try to enter a sleep state.
Parameters

| in  | mp          | pointer to the mutex_t structure |

Returns

The operation status.

Return values

| true | if the mutex has been successfully acquired |
| false | if the lock attempt failed. |

Function Class:

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

Here is the call graph for this function:

```
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

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>mp</td>
<td>pointer to the mutex_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:

7.27.4.7 chMtxUnlockS()

```c
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

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:

7.27.4.8 chMtxUnlockAllS()

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:

```
7.27.4.9 chMtxUnlockAll()

void chMtxUnlockAll ()
    void

Unlocks all mutexes owned by the invoking thread.

Postcondition

The stack of owned mutexes is emptied and all the found mutexes are unlocked.
Note

This function is **MUCH MORE** efficient than releasing the mutexes one by one and not just because the call overhead, this function does not have any overhead related to the priority inheritance mechanism.

Function Class:

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

Here is the call graph for this function:

```
chMtxUnlockAll
  ↓
  chMtxUnlockAllS
  ↓
  chSchReadyS
  ↓
  chSchRescheduleS
  ↓
  chThdGetSelfX
  ↓
  __dbg_check_lock
  ↓
  __stats_start_measure
  ↓
  _crit_thd
  ↓
  chMtxQueueNotEmptyS
  ↓
  ch_queue_fifo_remove
  ↓
  chSysLock
  ↓
  __stats_stop_measure
  ↓
  _crit_thd
  ↓
  chSysUnlock
  ↓
  __dbg_check_unlock
```

7.27.4.10  chMtxQueueNotEmptyS()

```c
static bool chMtxQueueNotEmptyS (mutex_t *mp) [inline], [static]
```

Returns `true` if the mutex queue contains at least a waiting thread.

**Parameters**

- **out** `mp` pointer to a `mutex_t` structure
Returns

The mutex queue status.

Function Class:

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

Here is the call graph for this function:

![Call graph]

### 7.27.4.11 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
```

### 7.27.4.12 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
```
7.28 Condition Variables

7.28.1 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_DECL(name) condition_variable_t name = __CONDVAR_DATA(name)
  
  Static condition variable initializer.

Typedefs

• typedef struct condition_variable condition_variable_t
  
  condition_variable_t structure.

Data Structures

• struct condition_variable
  
  condition_variable_t structure.

Functions

• void chCondObjectInit (condition_variable_t *cp)
  
  Initializes 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 chCondBroadcastI (condition_variable_t *cp)
  
  Signals all threads that are waiting on the condition variable.

• msg_t chCondWait (condition_variable_t *cp)
  
  Waits on the condition variable releasing the mutex lock.

• msg_t chCondWaitS (condition_variable_t *cp)
  
  Waits on the condition variable releasing the mutex lock.

• msg_t chCondWaitTimeout (condition_variable_t *cp, sysinterval_t timeout)
  
  Waits on the condition variable releasing the mutex lock.

• msg_t chCondWaitTimeoutS (condition_variable_t *cp, sysinterval_t timeout)
  
  Waits on the condition variable releasing the mutex lock.
7.28.2 Macro Definition Documentation

7.28.2.1 __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 |

7.28.2.2 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 |

7.28.3 Typedef Documentation

7.28.3.1 condition_variable_t

typedef struct condition_variable condition_variable_t

condition_variable_t structure.

7.28.4 Function Documentation
7.28.4.1 chCondObjectInit()

```c
void chCondObjectInit (
    condition_variable_t * cp )
```

Initializes a `condition_variable_t` structure.

**Parameters**

```
out cp pointer to a condition_variable_t structure
```

**Function Class:**

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

Here is the call graph for this function:

```
chCondObjectInit -> ch_queue_init
```

7.28.4.2 chCondSignal()

```c
void chCondSignal ( 
    condition_variable_t * cp )
```

Signals one thread that is waiting on the condition variable.

**Parameters**

```
in cp pointer to the condition_variable_t structure
```
Function Class:

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

Here is the call graph for this function:

7.28.4.3 chCondSignalI()

void chCondSignalI ( 
  condition_variable_t * cp )

Signals one thread that is waiting on the condition variable.

Postcondition

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

Parameters

| in | cp | pointer to the condition_variable_t structure |
Function Class:

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

Here is the call graph for this function:

```
<table>
<thead>
<tr>
<th>chCondSignalI</th>
</tr>
</thead>
<tbody>
<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>
```

### 7.28.4.4 chCondBroadcast()

```c
void chCondBroadcast (  
  condition_variable_t * cp )
```

Signals all threads that are waiting on the condition variable.

**Parameters**

| `in cp` | pointer to the `condition_variable_t` structure |
Function Class:

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

Here is the call graph for this function:

7.28.4.5 chCondBroadcastI()

void chCondBroadcastI (condition_variable_t * cp )

Signals all threads that are waiting on the condition variable.

Postcondition

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

Parameters

| in | cp | pointer to the condition_variable_t structure |
Function Class:

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

Here is the call graph for this function:

```
chCondBroadcastI
  |                     |                     |
  |                     |                     |
  v                     v                     v
ch_queue_fifo_remove  ch_queue_notempty  chDbgCheckClass
  |                     |                     |
  |                     |                     |
  v                     v                     v
chSchReadyI  __sch_ready_behind  chSysNotifyInstance
  |                     |                     |
  |                     |                     |
  v                     v                     v
chSysHalt
```

7.28.4.6 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:

```
7.28.4.7 chCondWaitS()

msg_t chCondWaitS {
    condition_variable_t * cp)
}
```

Waits on the condition variable releasing the mutex lock.

Releases the currently owned mutex, waits on the condition variable, and finally acquires the mutex again. All the sequence is performed atomically.

Precondition

The invoking thread must have at least one owned mutex.

Parameters

| in | cp | pointer to the condition_variable_t structure |
Returns

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

Return values

| MSG_OK          | if the condition variable has been signaled using chCondSignal(). |
| MSG_RESET       | if the condition variable has been signaled using chCondBroadcast(). |

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](ch_graph.png)

7.28.4.8 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.
Parameters

<table>
<thead>
<tr>
<th>Parameter</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>
<tr>
<td><code>in timeout</code></td>
<td>the number of ticks before the operation timeouts, the special values are handled as follow:</td>
</tr>
<tr>
<td>- <code>TIME_INFINITE</code></td>
<td>no timeout.</td>
</tr>
<tr>
<td>- <code>TIME_IMMEDIATE</code></td>
<td>this value is not allowed.</td>
</tr>
</tbody>
</table>

Returns

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

Return values

| MSG_OK | if the condition variable has been signaled using `chCondSignal()`. |
| MSG_RESET | if the condition variable has been signaled using `chCondBroadcast()`. |
| MSG_TIMEOUT | if the condition variable has not been signaled within the specified timeout. |

Function Class:

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

Here is the call graph for this function:

![Call Graph for chCondWaitTimeout](call_graph.png)
7.28 Condition Variables

7.28.4.9 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 <code>condition_variable_t</code> 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></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <code>TIME_INFINITE</code> no timeout.</td>
</tr>
<tr>
<td></td>
<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

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></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:

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

Here is the call graph for this function:
7.29 Event Flags

7.29.1 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 EVENTSOURCE_DECL(name)** event_source_t name = __EVENTSOURCE_DATA(name)
  
  Static event source initializer.
Typedefs

- typedef struct event_source event_source_t
  
  Event Source structure.

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

Data Structures

- struct event_listener
  
  Event Listener structure.

- struct event_source
  
  Event Source structure.

Functions

- void 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 chEvtSignall (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)
7.29 Event Flags

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.

7.29.2 Macro Definition Documentation

7.29.2.1 ALL_EVENTS

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

7.29.2.2 EVENT_MASK

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

7.29.2.3 __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.

ChibiOS/RT
7.29.2.4 EVENTSOURCE_DECL

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

Static event source initializer.

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

Parameters

| name | the name of the event source variable |

7.29.3 Typedef Documentation

7.29.3.1 event_source_t

typedef struct event_source event_source_t

Event Source structure.

7.29.3.2 evhandler_t

typedef void(* evhandler_t) (eventid_t id)

Event Handler callback function.

7.29.4 Function Documentation
7.29.4.1 chEvtRegisterMaskWithFlagsI()

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

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

Function Class:

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

Here is the call graph for this function:

7.29.4.2 chEvtRegisterMaskWithFlags()

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>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>elp</code></td>
<td>pointer to the <code>event_listener_t</code> structure</td>
</tr>
<tr>
<td><code>events</code></td>
<td>events to be ORed to the thread when the event source is broadcasted</td>
</tr>
<tr>
<td><code>wflags</code></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:

```
chEvtRegisterMaskWithFlags
    chEvtRegisterMaskWithFlagsI
        chThdSelfSetFX
            __dbg_check_lock
                __stats_start_measure
                    __crit_thd
                        __dbg_check_unlock
                            __stats_stop_measure
                                __crit_thd
                                    chSysUnlock
                                        chSysLock
                                            chEvtRegisterMaskWithFlags
```

### 7.29.4.3 chEvtUnregister()

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

Unregisters an Event Listener from its Event Source.

Note

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

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

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

Function Class:

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

Here is the call graph for this function:

```
+-----------------+                  +-----------------+                  +-----------------+
| chEvtUnregister |                  | chSysLock       |                  | __dbg_check_lock |
|                 |                  |                 |                  |                  |
| chEvtGetAndClearEventsI() |                  | _crit_thd       |                  |                  |
|                 |                  | __stats_start_measure |              |                  |
|                 |                  | __dbg_check_unlock |                  |                  |
|                 |                  |                  | __stats_stop_measure | _crit_thd       |
| chSysUnlock     |                  |                 |                  |                  |
```

7.29.4.4 chEvtGetAndClearEventsI()

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

Clears the pending events specified in the events mask.

Parameters

| in  | events | the events to be cleared |

Returns

The mask of pending events that were cleared.

Function Class:

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

7.29.4.5 chEvtGetAndClearEvents()

`eventmask_t chEvtGetAndClearEvents (`
`eventmask_t events)`

Clears the pending events specified in the events mask.

Parameters

| in  | events | the events to be cleared |

Returns

The mask of pending events that were cleared.
Function Class:
Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

7.29.4.6 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
- **in events** the events to be added

Returns
- The mask of currently pending events.
Function Class:

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

Here is the call graph for this function:

![Call Graph](image)

7.29.4.7 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.
7.29 Event Flags

Function 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
```

7.29.4.8 chEvtGetAndClearFlags()

```
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:

```
chEvtGetAndClearFlags
<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>___</td>
<td>___</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

7.29.4.9 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

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>in</strong> tp</td>
<td>the thread to be signaled</td>
</tr>
<tr>
<td><strong>in</strong> 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:

7.29.4.10  chEvtSignal()

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></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.
Here is the call graph for this function:

```
chEvtSignal
chEvtSignalI
chSchRescheduleS
chSysLock
chSysUnlock
chDbgCheckClassI
chSchReadyI
__sch_reschedule_ahead
chDbgCheckClassS
__dbg_check_lock
__stats_start_measure
_crit_thd
__dbg_check_unlock
__stats_stop_measure
_crit_thd
```

7.29.4.11 \texttt{chEvtBroadcastFlagsI()}

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

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

This function variants ORs the specified event flags to all the threads registered on the \texttt{event_source_t} in addition to the event flags specified by the threads themselves in the \texttt{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>\texttt{esp}</td>
<td>pointer to the \texttt{event_source_t} structure</td>
</tr>
<tr>
<td>\texttt{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:

```
<table>
<thead>
<tr>
<th>chEvtBroadcastFlags</th>
<th>chEvtSignal</th>
<th>chDbgCheckClass</th>
<th>chSysHalt</th>
</tr>
</thead>
<tbody>
<tr>
<td>__sch_reschedule_ahead</td>
<td>__dbg_check_lock</td>
<td>__stats_start_measure</td>
<td>__dbg_check_unlock</td>
</tr>
<tr>
<td>__stats_stop_measure</td>
<td>_crit_thd</td>
<td>chEvtBroadcastFlags</td>
<td>chEvtBroadcastFlags</td>
</tr>
<tr>
<td>chSchReady</td>
<td>chDbgCheckClassI</td>
<td>chEvtSignal</td>
<td>chEvtBroadcastFlags</td>
</tr>
<tr>
<td>chDbgCheckClassS</td>
<td>chEvtBroadcastFlags</td>
<td>chDbgCheckClassI</td>
<td>chEvtBroadcastFlags</td>
</tr>
<tr>
<td>chSysLock</td>
<td>chEvtSignal</td>
<td>chDbgCheckClassI</td>
<td>chEvtBroadcastFlags</td>
</tr>
<tr>
<td>chSysUnlock</td>
<td>chEvtBroadcastFlags</td>
<td>chDbgCheckClassI</td>
<td>chEvtBroadcastFlags</td>
</tr>
</tbody>
</table>
```

### 7.29.4.12 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**

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

**Function Class:**

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

Here is the call graph for this function:
7.29.4.13 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>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td><code>events</code></td>
<td>mask of events to be dispatched</td>
</tr>
<tr>
<td>in</td>
<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.

7.29.4.14 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>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td><code>events</code></td>
<td>events that the function should wait for, ALL_EVENTS 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:

```
chEvtWaitOne
  chSchGoSleepS
  chSysLock
  chSysUnlock
  chThdGetSelfX
  ch_pqueue_remove_highest
  chDbgCheckClassS
  chSysGetIdleThreadX
  __dbg_check_lock
  __stats_start_measure
  _crit_thd
  __dbg_check_unlock
  __stats_stop_measure
  _crit_thd
```

7.29.4.15 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 Image]

### 7.29.4.16 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.
7.29 Event Flags

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

Here is the call graph for this function:

7.29.4.17 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>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>
<tr>
<td><code>timeout</code></td>
<td>the number of ticks before the operation timeouts, the following special values are allowed:</td>
</tr>
<tr>
<td></td>
<td>• <code>TIME_IMMEDIATE</code> immediate timeout.</td>
</tr>
<tr>
<td></td>
<td>• <code>TIME_INFINITE</code> no timeout.</td>
</tr>
</tbody>
</table>
Returns

The mask of the lowest event id served and cleared.

Return values

0 if the operation has timed out.

Function Class:

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

Here is the call graph for this function:

7.29.4.18  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
7.29 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:

7.29.4.19 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>in</th>
<th>events</th>
<th>events that the function should wait for, <code>ALL_EVENTS</code> requires all the events</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td><code>timeout</code></td>
<td>the number of ticks before the operation timeouts, the following special values are allowed:</td>
</tr>
</tbody>
</table>

- **TIME_IMMEDIATE** immediate timeout.
- **TIME_INFINITE** no timeout.
Returns

The mask of the served and cleared events.

Return values

0 if the operation has timed out.

Function Class:

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

Here is the call graph for this function:

7.29.4.20 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 |
7.29 Event Flags

Function Class:

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

7.29.4.21 chEvtRegisterMask()

static void chEvtRegisterMask (  
    event_source_t * esp,  
    event_listener_t * elp,  
    eventmask_t events ) [inline], [static]

 Registers an Event Listener on an Event Source.

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

Note

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

Parameters

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

Function Class:

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

Here is the call graph for this function:

7.29.4.22 chEvtRegister()

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

Function Class:

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

Here is the call graph for this function:

```
chEvtRegister -> chEvtRegisterMask -> chEvtRegisterMaskWithFlags
```

7.29.4.23 chEvtIsListeningI()

Verifies if there is at least one event_listener_t registered.

Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>esp</td>
</tr>
</tbody>
</table>

Returns

The event source status.
7.29 Event Flags

Function Class:

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

7.29.4.24 chEvtBroadcast()

static void chEvtBroadcast ( 
    event_source_t ∗ esp ) [inline], [static]

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

Parameters

| in | esp | pointer to the event_source_t structure |

Function Class:

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

Here is the call graph for this function:

7.29.4.25 chEvtBroadcastI()

static void chEvtBroadcastI ( 
    event_source_t ∗ esp ) [inline], [static]

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

Postcondition

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

\[
\text{in} \quad \text{esp} \quad \text{pointer to the event\_source\_t structure}
\]

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:

![Call Graph](image)

### 7.29.4.26 chEvtAddEventsI()

```c
static eventmask_t chEvtAddEventsI (eventmask_t events) \{ \text{inline}, \text{static} \}
```

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

Parameters

\[
\text{in} \quad \text{events} \quad \text{the events to be added}
\]

Returns

The mask of currently pending events.

Function Class:

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

### 7.29.4.27 chEvtGetEventsX()

```c
static eventmask_t chEvtGetEventsX (void) \{ \text{inline}, \text{static} \}
```

Returns the events mask.

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

The pending events mask.

Function Class:

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

7.30.1 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 than `msg_t`. Messages are usually processed in FIFO order but it is possible to process them in priority order by enabling the `CH_CFG_USE_MESSAGES_PRIORITY` option in `chconf.h`.

Precondition

In order to use the message APIs the `CH_CFG_USE_MESSAGES` option must be enabled in `chconf.h`.

Postcondition

Enabling messages requires 6-12 (depending on the architecture) extra bytes in the `thread_t` structure.

Functions

- `msg_t chMsgSend (thread_t *tp, msg_t msg)`
  Sends a message to the specified thread.
- `thread_t * chMsgWait (void)`
  Suspends the thread and waits for an incoming message.
- `thread_t * chMsgWaitTimeout (sysinterval_t timeout)`
  Suspends the thread and waits for an incoming message or a timeout to occur.
- `thread_t * chMsgPoll (void)`
  Poll to check for an incoming message.
- `void chMsgRelease (thread_t *tp, msg_t msg)`
  Releases a sender thread specifying a response message.
- `static thread_t * chMsgWait (void)`
  Suspends the thread and waits for an incoming message.
- `static thread_t * chMsgWaitTimeout (sysinterval_t timeout)`
  Suspends the thread and waits for an incoming message or a timeout to occur.
- `static thread_t * chMsgPoll (void)`
  Poll to check for an incoming message.
- `static bool chMsgIsPending (thread_t *tp)`
  Evaluates to `true` if the thread has pending messages.
- `static msg_t chMsgGet (thread_t *tp)`
  Returns the message carried by the specified thread.
- `static void chMsgReleaseS (thread_t *tp, msg_t msg)`
  Releases the thread waiting on top of the messages queue.
7.30.2 Function Documentation

7.30.2.1 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:

```
\begin{tikzpicture}
    \node [circle, draw] (chMsgSend) {chMsgSend};
    \node [circle, draw, above of=chMsgSend] (chSysLock) {chSysLock};
    \node [circle, draw, below of=chMsgSend] (chThdGetSelfX) {chThdGetSelfX};
    \node [circle, draw, right of=chSysLock] (__dbg_check_lock) {__dbg_check_lock};
    \node [circle, draw, right of=chThdGetSelfX] (__stats_start_measure) {__stats_start_measure};
    \node [circle, draw, right of=__stats_start_measure] (__crit_thd) {__crit_thd};

    \draw [->] (chMsgSend) -- (chSysLock);
    \draw [->] (chMsgSend) -- (chThdGetSelfX);
    \draw [->] (chSysLock) -- (__dbg_check_lock);
    \draw [->] (chThdGetSelfX) -- (__stats_start_measure);
    \draw [->] (__stats_start_measure) -- (__crit_thd);
\end{tikzpicture}
```

7.30.2.2 chMsgWaitS()

```c
thread_t * chMsgWaitS(
    void )
```

Suspends the thread and waits for an incoming message.
Postcondition

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

Note

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

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

Returns

A pointer to the thread carrying the message.

Function Class:

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

Here is the call graph for this function:

```
chMsgWaitS
ch_queue_fifo_remove
chDbgCheckClassS
chMsgIsPendingI
chSchGoSleepS
chThdGetSelfX
chSysHalt
chDbgCheckClassI
ch_pqueue_remove_highest
chSysGetIdleThreadX
```

7.30.2.3 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.
7.30 Synchronous Messages

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:

---

7.30.2.4 chMsgPollS()

```c
thread_t * chMsgPollS ( void )
```

Poll to check for an incoming message.

Postcondition

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

Note

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

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

Returns

Result of the poll.
Return values

NULL if no incoming message waiting.

Function Class:

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

Here is the call graph for this function:

```
ch_msg_poll
    |  \   |
    v   v
ch_queue_fifo_remove
    |     |
    \   |
        v
ch_msg_is_pending
    |  \   |
    v     v
ch_dbg_check_class
    |     |
    \   |
          v
ch_thd_get_self
```

### 7.30.2.5 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]

### 7.30.2.6 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.
Function Class:

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

Here is the call graph for this function:

```
7.30.2.7 chMsgWaitTimeout()

static thread_t* chMsgWaitTimeout (sysinterval_t timeout) [inline], [static]

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

Postcondition

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

Note

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

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

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

### Returns

A pointer to the thread carrying the message.

### Return values

\textbf{NULL} if a timeout occurred.

### Function Class:

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

Here is the call graph for this function:

![Call graph diagram]

#### 7.30.2.8 chMsgPoll()

```c
static thread_t* chMsgPoll (void) [inline], [static]
```

Poll to check for an incoming message.
Postcondition

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

Note

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

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

Returns

A pointer to the thread carrying the message.

Return values

- **NULL** if no incoming message waiting.

Function Class:

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

Here is the call graph for this function:

```
ch_MsgPoll     ch_MsgPollS
ch.SysLock     ch_MsgIsPending
chQueue_fifoh_remove
```
7.30 Synchronous Messages

7.30.2.9 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:

7.30.2.10 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

ChibiOS/RT
Returns

The message carried by the sender.

Function Class:

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

7.30.2.11  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

<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:

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

Here is the call graph for this function:
7.31 Dynamic Threads

7.31.1 Detailed Description

Dynamic threads related APIs and services.

Functions

- \( \text{thread_t} * \) 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.

- \( \text{thread_t} * \) chThdCreateFromMemoryPool (memory_pool_t *mp, const char *name, tprio_t prio, tfunc_t pf, void *arg)
  
  Creates a new thread allocating the memory from the specified memory pool.

7.31.2 Function Documentation

7.31.2.1 chThdCreateFromHeap()

\[
\text{thread_t} * \text{chThdCreateFromHeap} (\text{memory_heap_t} * \text{heapp}, \text{size_t} \text{ size}, \text{const char} * \text{name}, \text{tprio_t} \text{ prio}, \text{tfunc_t} \text{ pf}, \text{void} * \text{arg})
\]

Creates a new thread allocating the memory from the heap.

Precondition

The configuration options \text{CH_CFG_USE_DYNAMIC} and \text{CH_CFG_USE_HEAP} must be enabled in order to use this function.

Note

A thread can terminate by calling \text{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 \text{chThdWait()} and then release the allocated memory.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>heapp</strong></td>
<td>heap from which allocate the memory or NULL for the default heap</td>
</tr>
<tr>
<td><strong>size</strong></td>
<td>size of the working area to be allocated</td>
</tr>
<tr>
<td><strong>name</strong></td>
<td>thread name</td>
</tr>
<tr>
<td><strong>prio</strong></td>
<td>the priority level for the new thread</td>
</tr>
<tr>
<td><strong>pf</strong></td>
<td>the thread function</td>
</tr>
<tr>
<td><strong>arg</strong></td>
<td>an argument passed to the thread function. It can be NULL.</td>
</tr>
</tbody>
</table>
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:

![Call Graph](attachment://call_graph.png)

### 7.31.2.2 chThdCreateFromMemoryPool()

```c
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 \texttt{CH\_CFG\_USE\_DYNAMIC} and \texttt{CH\_CFG\_USE\_MEMPOOLS} must be enabled in order to use this function.

The pool must be initialized to contain only objects with alignment \texttt{PORT\_WORKING\_AREA\_ALIGN}.

Note

A thread can terminate by calling \texttt{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 \texttt{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 \texttt{NULL}.</td>
</tr>
</tbody>
</table>

Returns

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

Return values

\textbf{NULL} if the memory pool is empty.

Function Class:

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

Here is the call graph for this function:
7.32 Registry

7.32.1 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.

7.32.2 Macro Definition Documentation

7.32.2.1 REG_HEADER

#define REG_HEADER(oip) (&ch_system.reglist.queue)

Access to the registry list header.

7.32.2.2 REG_REMOVE

#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 |

ChibiOS/RT
7.32.2.3 REG_INSERT

#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>

7.32.3 Function Documentation

7.32.3.1 chRegFirstThread()

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:

```
chRegFirstThread
chSysLock
chSysUnlock
__dbg_check_lock
__stats_start_measure
_crit_thd
__dbg_check_unlock
__stats_stop_measure
_crit_thd
```

7.32.3.2 chRegNextThread()

```c
thread_t * chRegNextThread ( thread_t * tp )
```

Returns the thread next to the specified one.

The reference counter of the specified thread is decremented and the reference counter of the returned thread is incremented.

Parameters

- `tp` pointer to the thread

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:

![Call Graph Image]

### 7.32.3.3 chRegFindThreadByName()

```c
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:

```
chRegFindThreadByName
chRegFirstThread
chRegGetThreadNameX
chRegNextThread
chSysLock
chSysUnlock
chThdRelease
```

### 7.32.3.4 chRegFindThreadByPointer()

```c
thread_t * chRegFindThreadByPointer ( thread_t * tp )
```

Confirms that a pointer is a valid thread pointer.

**Note**

The reference counter of the found thread is increased by one so it cannot be disposed incidentally after the pointer has been returned.

**Parameters**

| 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:

```
chRegFindThreadByPointer
chRegFirstThread
chRegNextThread
chSysLock
chSysUnlock
chThdRelease
chThdRelease
```

### 7.32.3.5 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:

```
chRegFindThreadByWorkingArea
chRegFirstThread
chRegNextThread
chThdGetWorkingAreaX
chSysLock
chSysUnlock
chThdRelease
```

### 7.32.3.6 __reg_object_init()

```c
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:

```
__reg_object_init
ch_queue_init
```
7.32.3.7 chRegSetThreadName()

```c
static void chRegSetThreadName (  
    const char * name ) [inline], [static]
```

Sets the current thread name.

**Precondition**

This function only stores the pointer to the name if the option `CH_CFG_USE_REGISTRY` is enabled else no action is performed.

**Parameters**

| in | name | thread name as a zero terminated string |

**Function Class:**

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

7.32.3.8 chRegGetThreadNameX()

```c
static const char* chRegGetThreadNameX (  
    thread_t * tp ) [inline], [static]
```

Returns the name of the specified thread.

**Precondition**

This function only returns the pointer to the name if the option `CH_CFG_USE_REGISTRY` is enabled else `NULL` is returned.

**Parameters**

| in | tp | pointer to the thread |

**Returns**

Thread name as a zero terminated string.

**Return values**

| NULL | if the thread name has not been set. |
7.32 Registry

7.32.3.9 chRegSetThreadNameX()

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></th>
<th>tp</th>
<th>pointer to the thread</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>name</td>
<td>thread name as a zero terminated string</td>
</tr>
</tbody>
</table>

Function Class:

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

7.33.1 Detailed Description

Modules

- Checks and Assertions
- Tracing
- Statistics
7.34 Checks and Assertions

7.34.1 Detailed Description

Debug APIs and services:

- Runtime system state and call protocol check. The following panic messages can be generated:
  - SV#1, misplaced \texttt{chSysDisable()}.
    - Called from an ISR.
    - Called from a critical zone.
  - SV#2, misplaced \texttt{chSysSuspend()}.
    - Called from an ISR.
    - Called from a critical zone.
  - SV#3, misplaced \texttt{chSysEnable()}.
    - Called from an ISR.
    - Called from a critical zone.
  - SV#4, misplaced \texttt{chSysLock()}.
    - Called from an ISR.
    - Called from a critical zone.
  - SV#5, misplaced \texttt{chSysUnlock()}.
    - Called from an ISR.
    - Not called from a critical zone.
  - SV#6, misplaced \texttt{chSysLockFromISR()}.
    - Not called from an ISR.
    - Called from a critical zone.
  - SV#7, misplaced \texttt{chSysUnlockFromISR()}.
    - Not called from an ISR.
    - Not called from a critical zone.
  - SV#8, misplaced \texttt{CH_IRQ_PROLOGUE()}.
    - Not called at ISR begin.
    - Called from a critical zone.
  - SV#9, misplaced \texttt{CH_IRQ_EPILOGUE()}.
    - \texttt{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.

\textbf{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.
7.34 Checks and Assertions

7.34.2 Macro Definition Documentation

7.34.2.1 CH_DBG_STACK_FILL_VALUE

#define CH_DBG_STACK_FILL_VALUE 0x55

Fill value for thread stack area in debug mode.

7.34.2.2 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/\n            chSysHalt(__func__);
        }
    }
} while (false)

Function parameters check.

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

Note

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

Parameters

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

Function Class:

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

7.34.2.3 chDbgAssert

#define chDbgAssert(c, r)
Condition assertion.

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

Note

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

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

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in <code>c</code></td>
<td>the condition to be verified to be true</td>
</tr>
<tr>
<td>in <code>r</code></td>
<td>a remark string</td>
</tr>
</tbody>
</table>

Function Class:

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

### 7.34.3 Typedef Documentation

#### 7.34.3.1 system_debug_t

```c
typedef struct ch_system_debug system_debug_t
```

System debug data structure.

### 7.34.4 Function Documentation
### 7.34.1 __dbg_check_disable()

```c
void __dbg_check_disable ( void )
```

Guard code for `chSysDisable()`.

**Function Class:**

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

Here is the call graph for this function:

![Call Graph](image)

### 7.34.2 __dbg_check_suspend()

```c
void __dbg_check_suspend ( void )
```

Guard code for `chSysSuspend()`.

**Function Class:**

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

Here is the call graph for this function:

![Call Graph](image)
7.34.4.3  __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:

![Call Graph for __dbg_check_enable()]

7.34.4.4  __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:

![Call Graph for __dbg_check_lock()]

7.34 Checks and Assertions

7.34.4.5 __dbg_check_unlock()

```c
void __dbg_check_unlock ( void )
```

Guard code for `chSysUnlock()`.

Function Class:

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

Here is the call graph for this function:

```
__dbg_check_unlock -> chSysHalt -> __trace_halt
```

7.34.4.6 __dbg_check_lock_from_isr()

```c
void __dbg_check_lock_from_isr ( void )
```

Guard code for `chSysLockFromIsr()`.

Function Class:

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

Here is the call graph for this function:

```
__dbg_check_lock_from_isr -> chSysHalt -> __trace_halt
```
7.34.4.7 __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 for __dbg_check_unlock_from_isr()](image)

7.34.4.8 __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 for __dbg_check_enter_isr()](image)
7.34 Checks and Assertions

7.34.4.9  __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:

[Call graph image]

7.34.4.10  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:

[Call graph image]
7.34.4.11 chDbgCheckClassS()

```c
void chDbgCheckClassS ( void )
```

S-class functions context check.

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

Function Class:

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

Here is the call graph for this function:

```
chDbgCheckClassS  chSysHalt  __trace_halt
```

7.34.4.12 __dbg_object_init()

```c
static void __dbg_object_init ( system_debug_t * sdp ) [inline], [static]
```

Debug support initialization.

Note

Internal use only.

Parameters

```
out  sdp  pointer to the system_debug_t structure
```

Function Class:

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

7.35.1 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 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.
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.

7.35.2 Macro Definition Documentation

7.35.2.1 CH_DBG_TRACE_MASK

#define CH_DBG_TRACE_MASK CH_DBG_TRACE_MASK_DISABLED
Trace buffer entries.

7.35.2.2 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.
7.35.3 Function Documentation

7.35.3.1 trace_next()

```c
static NOINLINE 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](chart)

7.35.3.2 __trace_object_init()

```c
void __trace_object_init ( trace_buffer_t * tbp )
```

Circular trace buffer initialization.

Note

Internal use only.

Parameters

- `tbp` pointer to the `trace_buffer_t` structure

Function Class:

Not an API, this function is for internal use only.
7.35.3.3 __trace_ready()

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

Inserts in the circular debug trace buffer a ready record.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>tp</code></td>
<td>the thread that just became ready</td>
</tr>
<tr>
<td><code>msg</code></td>
<td>the thread ready message</td>
</tr>
</tbody>
</table>

**Function Class:**

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

7.35.3.4 __trace_switch()

```c
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>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ntp</code></td>
<td>the thread being switched in</td>
</tr>
<tr>
<td><code>otp</code></td>
<td>the thread being switched out</td>
</tr>
</tbody>
</table>

**Function Class:**

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

7.35.3.5 __trace_isr_enter()

```c
void __trace_isr_enter ( 
    const char * isr )
```

Inserts in the circular debug trace buffer an ISR-enter record.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>isr</code></td>
<td>name of the isr</td>
</tr>
</tbody>
</table>
Function Class:
Not an API, this function is for internal use only.

7.35.3.6 __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.

7.35.3.7 __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.

7.35.3.8 chTraceWriteI()

```c
void chTraceWriteI (void * up1, void * up2 )
```

Adds an user trace record to the trace buffer.
Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>$up1$</th>
<th>user parameter 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>$up2$</td>
<td>user parameter 2</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:

```
chTraceWriteI -> chDbgCheckClassI -> chSysHalt
```

### 7.35.3.9 chTraceWrite()

```c
void chTraceWrite(
    void * $up1$,
    void * $up2$
)
```

Adds an user trace record to the trace buffer.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>$up1$</th>
<th>user parameter 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>$up2$</td>
<td>user parameter 2</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:

```
chTraceWrite
    |                        __dbg_check_lock
    v                        __stats_start_measure
chSysLock
    |                          __crit_thd
    v                          __dbg_check_unlock
    v                          __stats_stop_measure
    v                          __crit_thd
chTraceWriteI
```

### 7.35.3.10 chTraceSuspendI()

```c
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:

![Call Graph](image)

7.35.3.11 chTraceSuspend()

```c
void chTraceSuspend (  
    uint16_t mask  )
```

Suspends one or more trace events.

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>mask</td>
<td>mask of the trace events to be suspended</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:

```
void chTraceResumeI ( uint16_t mask )

Resumes one or more trace events.

Parameters

- **mask** mask of the trace events to be resumed

Function Class:

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

Here is the call graph for this function:

```


```


7.35.3.13 chTraceResume()

```c
void chTraceResume (       
    uint16_t mask )
```

Resumes one or more trace events.

**Parameters**

| in | mask | mask of the trace events to be resumed |

**Function Class:**

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

Here is the call graph for this function:
7.36 Statistics

7.36.1 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.

7.36.2 Function Documentation

7.36.2.1 __stats_increase_irq()

void __stats_increase_irq (void)
  
  void

Increases the IRQ counter.

7.36.2.2 __stats_ctxswc()

void __stats_ctxswc (thread_t *ntp,
                        thread_t *otp)

Updates context switch related statistics.
Parameters

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

Here is the call graph for this function:

![Call Graph](call_graph.png)

### 7.36.2.3 __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:

![Call Graph](call_graph.png)

### 7.36.2.4 __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:

```c
__stats_stop_measure_crit_thd
```

7.36.2.5  __stats_start_measure_crit_isr()

```c
void __stats_start_measure_crit_isr ( void )
```

Starts the measurement of an ISR critical zone.

Here is the call graph for this function:

```c
__stats_start_measure_crit_isr
```

7.36.2.6  __stats_stop_measure_crit_isr()

```c
void __stats_stop_measure_crit_isr ( void )
```

Stops the measurement of an ISR critical zone.

Here is the call graph for this function:

```c
__stats_stop_measure_crit_isr
```
7.36.2.7 __stats_object_init()

```c
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:
7.37 OS Library

7.37.1 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
7.38 Version Numbers and Identification

7.38.1 Detailed Description

OS Library related info.

Macros

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

ChibiOS/LIB version identification

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

Functions

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

7.38.2 Macro Definition Documentation

7.38.2.1 __CHIBIOS_OSLIB__

`#define __CHIBIOS_OSLIB__`
ChibiOS/LIB identification macro.

7.38.2.2 CH_OSLIB_STABLE

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

#define CH_OSLIB_VERSION "1.3.0"

OS Library version string.

7.38.2.4 CH_OSLIB_MAJOR

#define CH_OSLIB_MAJOR 1

OS Library version major number.

7.38.2.5 CH_OSLIB_MINOR

#define CH_OSLIB_MINOR 3

OS Library version minor number.

7.38.2.6 CH_OSLIB_PATCH

#define CH_OSLIB_PATCH 0

OS Library version patch number.

7.38.3 Function Documentation
7.38.3.1 __oslib_init()

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

Initialization of all library modules.

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

Here is the call graph for this function:
7.39 Synchronization

7.39.1 Detailed Description

Synchronization services.

Modules

- Binary Semaphores
- Mailboxes
- Pipes
- Delegate Threads
- Jobs Queues
## 7.40 Binary Semaphores

### 7.40.1 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.

### 7.40.2 Macro Definition Documentation
### 7.40.2.1 __BSEMAPHORE_DATA

```c
#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>

### 7.40.2.2 BSEMAPHORE_DECL

```c
#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>

### 7.40.3 Typedef Documentation

#### 7.40.3.1 binary_semaphore_t

```c
typedef struct ch_binary_semaphore binary_semaphore_t
```

Binary semaphore type.

### 7.40.4 Function Documentation
7.40.4.1 chBSemObjectInit()

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

Initializes a binary semaphore.

**Parameters**

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

**Function Class:**

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

Here is the call graph for this function:

![Call Graph](#)

7.40.4.2 chBSemWait()

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

Wait operation on the binary semaphore.

**Parameters**

| in  | bsp        | pointer to a binary_semaphore_t structure |

**Returns**

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

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

Function Class:

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

Here is the call graph for this function:

```
chBSemWait  chSemWait       chSemWaitS
   |             |               |
   v             v               v
chSysLock  chSysUnlock
```

7.40.4.3 chBSemWaitS()

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

Wait operation on the binary semaphore.

Parameters

| in | bsp | pointer to a binary_semaphore_t structure |

Returns

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

Return values

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

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

Here is the call graph for this function:

```
chBSemWaitS -> chDbgCheckClassS -> chSysHalt
  |       |                   |
  v       v                   v
chSemWaitS -> ch_queue_isempty
        |                   |
        v                   v
ch_queue_notempty -> chThdGetSelfX
```

7.40.4.4  **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>• <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 binary semaphore has been successfully taken. |
| MSG_RESET  | if the binary semaphore has been reset using bsemReset(). |
| MSG_TIMEOUT| if the binary semaphore has not been signaled or reset within the specified timeout. |
Function Class:

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

Here is the call graph for this function:

![Call Graph](image)

### 7.40.4.5 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>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>bsp</code></td>
<td>Pointer to a <code>binary_semaphore_t</code> structure</td>
</tr>
<tr>
<td><code>timeout</code></td>
<td>The number of ticks before the operation timeouts, the following special values are allowed:</td>
</tr>
<tr>
<td></td>
<td>- <code>TIME_IMMEDIATE</code> immediate timeout.</td>
</tr>
<tr>
<td></td>
<td>- <code>TIME_INFINITE</code> no timeout.</td>
</tr>
</tbody>
</table>

**Returns**

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

**Return values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MSG_OK</strong></td>
<td>If the binary semaphore has been successfully taken.</td>
</tr>
<tr>
<td><strong>MSG_RESET</strong></td>
<td>If the binary semaphore has been reset using <code>bsemReset()</code>.</td>
</tr>
<tr>
<td><strong>MSG_TIMEOUT</strong></td>
<td>If the binary semaphore has not been signaled or reset within the specified timeout.</td>
</tr>
</tbody>
</table>
Function Class:

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

Here is the call graph for this function:

```
chBSemWaitTimeout
  chSemWaitTimeout
  chSemWaitTimeoutS
  chSysLock
  chSysUnlock
```

7.40.4.6 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>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>bsp</code></td>
<td>pointer to a <code>binary_semaphore_t</code> structure</td>
</tr>
<tr>
<td><code>taken</code></td>
<td>new state of the binary semaphore</td>
</tr>
<tr>
<td></td>
<td>• <code>false</code>, the new state is not taken.</td>
</tr>
<tr>
<td></td>
<td>• <code>true</code>, the new state is taken.</td>
</tr>
</tbody>
</table>
7.40 Binary Semaphores

Function Class:

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

Here is the call graph for this function:

```
chBSemResetI chDbgCheckClassI
chSemResetI
chSysHalt
chSemResetWithMessageI
```

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

```
chSemReset chBSemReset chSemResetWithMessage
```

ChibiOS/RT
7.40.4.8  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

in  bsp  pointer to a binary_semaphore_t structure

Function Class:

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

Here is the call graph for this function:

7.40.4.9  chBSemSignal()

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

Performs a signal operation on a binary semaphore.

Parameters

in  bsp  pointer to a binary_semaphore_t structure
7.40 Binary 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:

```
chBSemSignal
chBSemSignalI
chSchRescheduleS
chSysLock
chSysUnlock
chDbgCheckClassI
chSemSignalI
__sch_reschedule_ahead
chDbgCheckClassS
__dbg_check_lock
__stats_start_measure
_crit_thd
__dbg_check_unlock
__stats_stop_measure
_crit_thd
```

7.40.4.10 chBSemGetStateI()

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

Returns the binary semaphore current state.

Parameters

| in  | bsp | pointer to a binary_semaphore_t structure |

Returns

The binary semaphore current state.

Return values

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

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

Here is the call graph for this function:
7.41 Mailboxes

7.41.1 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 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 chMBGetUsedCount (const mailbox_t *mbp)
  
  Returns the number of used message slots into a mailbox.

- static size_t chMBGetFreeCount (const mailbox_t *mbp)
  
  Returns the number of free message slots into a mailbox.

- static msg_t chMBPeekI (const mailbox_t *mbp)
  
  Returns the next message in the queue without removing it.

- static void chMBResumeX (mailbox_t *mbp)
  
  Terminates the reset state.

7.41.2 Macro Definition Documentation
7.41 Mailboxes

7.41.2.1 __MAILBOX_DATA

#define __MAILBOX_DATA(
    name,
    buffer,
    size )

Value:
{
    (msg_t *)buffer,
    (msg_t *)buffer + size,
    (msg_t *)buffer,
    (size_t)0,
    false,
    __THREADS_QUEUE_DATA(name.qw),
    __THREADS_QUEUE_DATA(name.qr),
}

Data part of a static mailbox initializer.

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

Parameters

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

7.41.2.2 MAILBOX_DECL

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

Static mailbox initializer.

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

Parameters

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

7.41.3 Function Documentation
### 7.41.3.1 chMBObjectInit()

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

Initializes a `mailbox_t` object.

**Parameters**

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

**Function Class:**

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

Here is the call graph for this function:

![Call Graph]

### 7.41.3.2 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>in</th>
<th>mbp</th>
<th>the pointer to an initialized <code>mailbox_t</code> object</th>
</tr>
</thead>
</table>

ChibiOS/RT
Function Class:

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

Here is the call graph for this function:

```
7.41.3.3 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:

```
chMBResetI
chDbgCheckClassI
chThdDequeueAllI
chSysHalt
ch_queue_notempty
chThdDoDequeueNextI
```

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

**Returns**

The operation status.

**Return values**

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

Function Class:

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

Here is the call graph for this function:

7.41.3.5 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>
<tr>
<td></td>
<td></td>
<td>• <code>TIME_IMMEDIATE</code> immediate timeout.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <code>TIME_INFINITE</code> no timeout.</td>
</tr>
</tbody>
</table>

Returns

The operation status.

Return values

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

Function Class:

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

Here is the call graph for this function:

```
7.41.3.6 chMBPostI()

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

The operation status.

Return values

<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 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]

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

The operation status.

Return values

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

Function Class:

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

Here is the call graph for this function:

7.41.3.8 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 an empty slot in the mailbox becomes available or the specified time runs out.

Parameters

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

Return values

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

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

Here is the call graph for this function:

```
7.41.3.9 chMBPostAheadI()

msg_t chMBPostAheadI (  
    mailbox_t * mbp,  
    msg_t msg  
)
```

Posts an high priority message into a mailbox.
This variant is non-blocking, the function returns a timeout condition if the queue is full.

Parameters

| in   | mbp  | the pointer to an initialized mailbox_t object |
| in   | msg  | the message to be posted on the mailbox       |
Returns

The operation status.

Return values

| MSG_OK     | if a message has been correctly posted. |
| MSG_RESET  | if the mailbox has been reset.          |
| MSG_TIMEOUT| if the mailbox is full and the message cannot be posted. |

Function Class:

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

Here is the call graph for this function:

```
7.41.3.10 chMBFetchTimeout()

msg_t chMBFetchTimeout (  
    mailbox_t * mbp,  
    msg_t * msgp,  
    sysinterval_t timeout )
```

Retrieves a message from a mailbox.

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

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>mbp</th>
<th>the pointer to an initialized <code>mailbox_t</code> object</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>msgp</td>
<td>pointer to a message variable for the received message</td>
</tr>
<tr>
<td>in</td>
<td>timeout</td>
<td>the number of ticks before the operation timeouts, the following special values are allowed:</td>
</tr>
</tbody>
</table>

- TIME_IMMEDIATE immediate timeout.
- TIME_INFINITY no timeout.
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:

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

Here is the call graph for this function:

```
7.41.3.11 chMBFetchTimeoutS()

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

Retrieves a message from a mailbox.

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

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

Returns

The operation status.

Return values

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

Function Class:

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

Here is the call graph for this function:

![Call Graph](image)

### 7.41.3.12 chMBFetchI()

```c
msg_t chMBFetchI (  
    mailbox_t * mbp,  
    msg_t * msgp  
)  
```

Retrieves a message from a mailbox.

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

<table>
<thead>
<tr>
<th>in</th>
<th>mbp</th>
<th>the pointer to an initialized <code>mailbox_t</code> object</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>msgp</td>
<td>pointer to a message variable for the received message</td>
</tr>
</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 mailbox is empty and a message cannot be fetched.</td>
</tr>
</tbody>
</table>

Function Class:

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

Here is the call graph for this function:

```
    chMBFetchI
      
      chMBGetSizeI
      
      chThdDequeueNextI
        
        chSysHalt
```

### 7.41.3.13 chMBGetSizeI()

```c
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 |

ChibiOS/RT
Returns

The size of the mailbox.

Function Class:

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

7.41.3.14 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:

7.41.3.15 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](image)

7.41.3.16 chMBPeekI()

```c
static msg_t chMBPeekI ( 
    const mailbox_t * mbp ) [inline], [static]
```

Returns the next message in the queue without removing it.

Precondition

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

Parameters

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

Returns

The next message in queue.
Function Class:

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

Here is the call graph for this function:

```
chMBPeekI  →  chDbgCheckClassI  →  chSysHalt
```

7.41.3.17  **chMBResumeX()**

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

Terminates the reset state.

**Parameters**

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

Function Class:

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

7.42.1 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.

7.42.2 Macro Definition Documentation
7.42.2.1 __PIPE_DATA

#define __PIPE_DATA(
    name,
    buffer,
    size )

Value:

\[
\begin{align*}
\text{(uint8_t *)} & \text{buffer}, \\
\text{(uint8_t *)} & \text{buffer} + \text{size}, \\
\text{(uint8_t *)} & \text{buffer}, \\
\text{(size_t)} & 0, \\
\text{false}, \\
\text{NULL}, \\
\text{NULL}, \\
\text{__MUTEX_DATA(name.cmtx)}, \\
\text{__MUTEX_DATA(name.wmtx)}, \\
\text{__MUTEX_DATA(name.rmtx)}, \\
\end{align*}
\]

Data part of a static pipe initializer.

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

Parameters

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

7.42.2.2 PIPE_DECL

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

Static pipe initializer.

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

Parameters

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

7.42.3 Function Documentation
7.42.3.1 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

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

Returns

The number of bytes effectively transferred.

Function Class:

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

7.42.3.2 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

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

Returns

The number of bytes effectively transferred.
Function Class:

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

### 7.42.3.3 chPipeObjectInit()

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

Initializes a `mailbox_t` object.

**Parameters**

| out | pp | the pointer to the `pipe_t` structure to be initialized |
| in  | buf | pointer to the pipe buffer as an array of `uint8_t` |
| in  | n   | number of elements in the buffer array |

Function Class:

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

### 7.42.3.4 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.
### 7.42.3.5 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>in</th>
<th>pp</th>
<th>the pointer to an initialized pipe_t object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>bp</td>
<td>pointer to the data buffer</td>
</tr>
<tr>
<td>in</td>
<td>n</td>
<td>the 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.

### 7.42.3.6 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>in</th>
<th>pp</th>
<th>the pointer to an initialized pipe_t object</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>bp</td>
<td>pointer to the data buffer</td>
</tr>
<tr>
<td>in</td>
<td>n</td>
<td>the number of bytes to be read, the value 0 is reserved</td>
</tr>
<tr>
<td>in</td>
<td>timeout</td>
<td>the number of ticks before the operation timeouts, the following special values are allowed:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 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.

### 7.42.3.7 chPipeGetSize()

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

Returns the pipe buffer size as number of bytes.

Parameters

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

Returns

The size of the pipe.

Function Class:

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

### 7.42.3.8 chPipeGetUsedCount()

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

Returns the number of used byte slots into a pipe.

Parameters

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

Returns

The number of queued bytes.
7.42 Pipes

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

7.42.3.9 chPipeGetFreeCount()

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

Returns the number of free byte slots into a pipe.

Parameters
in  pp  the pointer to an initialized pipe_t object

Returns
The number of empty byte slots.

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

Here is the call graph for this function:

7.42.3.10 chPipeResume()

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

Terminates the reset state.
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>the pointer to an initialized pipe_t 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.
7.43 Delegate Threads

7.43.1 Detailed Description

**Typedefs**

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

**Functions**

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

7.43.2.1 delegate_veneer_t

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

7.43.2.2 delegate_fn0_t

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

7.43.2.3 delegate_fn1_t

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

7.43.2.4 delegate_fn2_t

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

7.43.2.5 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.

7.43.2.6 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.

7.43.3 Function Documentation

7.43.3.1 __ch_delegate_fn0()

msg_t __ch_delegate_fn0 {
    va_list * argsp
}
Veneer for functions with no parameters.
7.43 Delegate Threads

Parameters

*in* `argsp` the list of arguments

Returns

The function return value.

### 7.43.3.2 __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.

### 7.43.3.3 __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.

### 7.43.3.4 __ch_delegate_fn3()

```c
msg_t __ch_delegate_fn3 {
    va_list * argsp
}
```
Veneer for functions with three parameters.
7.43 Delegate Threads

Parameters

| in | argsp | the list of arguments |

Returns

The function return value.

7.43.3.5 __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.

7.43.3.6 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:

```
chDelegateCallVeneer  chMsgSend    chSysLock
                        chThdGetSelfX
```

7.43.3.7  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:

```
chDelegateDispatch  chMsgGet    chMsgReleaseS
                    chMsgRelease  chSysLock
                    chMsgWait    chSysUnlock
```

```
chDelegateDispatch  chMsgGet    chMsgReleaseS
                    chMsgRelease  chSysLock
                    chMsgWait    chSysUnlock
```
7.43 Delegate Threads

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

**Returns**

The function outcome.

**Return values**

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

**Function Class:**

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

Here is the call graph for this function:

![Call Graph](image_url)
7.43.9 chDelegateCallDirect0()

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

Direct call to a function with no parameters.

**Note**

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

**Parameters**

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

**Returns**

The function return value as a `msg_t`.

Here is the call graph for this function:

![Call Graph](image)

7.43.10 chDelegateCallDirect1()

```c
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:

```plaintext
chDelegateCallDirect1
  __ch_delegate_fn1
  chDelegateCallVeneer chMsgSend
```

### 7.43.3.11 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:

![Call Graph](call_graph.png)

### 7.43.3.12 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**

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

The function return value as a msg_t.

Here is the call graph for this function:

```
    __ch_delegate_fn3
       |                 |              |
       |                 |              |
       |                 |              |
       v                 v              v
chDelegateCallDirect3 --> chDelegateCallVeneer --> chMsgSend
```

### 7.43.3.13 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.

| 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 |
| in | p4    | parameter 4 passed as a msg_t |
Returns

The function return value as a `msg_t`.

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

7.44.1 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.
7.44.2 Macro Definition Documentation

7.44.2.1 MSG_JOB_NULL

#define MSG_JOB_NULL ((msg_t)-2)

Dispatcher return code in case of a JOB_NULL has been received.

7.44.3 Typedef Documentation

7.44.3.1 jobs_queue_t

typedef struct ch_jobs_queue jobs_queue_t

Type of a jobs queue.

7.44.3.2 job_function_t

typedef void(* job_function_t) (void *arg)

Type of a job function.

7.44.3.3 job_descriptor_t

typedef struct ch_job_descriptor job_descriptor_t

Type of a job descriptor.

7.44.4 Function Documentation

7.44.4.1 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.
7.44 Jobs Queues

Parameters

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

Function Class:

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

Here is the call graph for this function:

```
chJobObjectInit
  |__________________________|
  |                          |
  | chGuardedPoolLoadArray   |
  |                        /|
  | chGuardedPoolAdd        |  \
  |________________________|  |
  |                           |
  | chJobObjectInit          |
  |________________________|
  |                          |
  | chGuardedPoolObjectInit   |
  |                        /|
  | chGuardedPoolObjectInitAligned |
  |________________________|
  |                          |
  | chMBObjectInit          |
  |________________________|
  |                          |
  | chThdQueueObjectInit    |
```

7.44.4.2 chJobGet()

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

Allocates a free job object.

Parameters

| in       | jqp                              | pointer to a jobs_queue_t structure |

Returns

The pointer to the allocated job object.
Function Class:

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

Here is the call graph for this function:

![Call Graph]

7.44.4.3 chJobGet()

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

Allocates a free job object.

**Parameters**

| in | jqp | pointer to a jobs_queue_t structure |

**Returns**

The pointer to the allocated job object.

**Return values**

| NULL | if a job object is not immediately available. |
7.44 Jobs Queues

Function 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
  chPoolAllocI
  chSemFastWaitI
  chSemGetCounterI
```

7.44.4.4 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>• 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:

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
                  chPoolAlloc
                  chSemWaitTimeoutS
```

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

**Returns**

The pointer to the allocated job object.

**Return values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NULL</td>
<td>if a job object is not available within the specified timeout.</td>
</tr>
</tbody>
</table>
Function Class:
Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

```
chJobGetTimeout -> chGuardedPoolAllocTimeout
|                   |                  |
|                   |                  |
|                   |                  |
|                   |                  |
|                   |                  |
|                   |                  |
```

7.44.4.6 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>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>in</code></td>
<td><code>jqp</code> pointer to a <code>jobs_queue_t</code> structure</td>
</tr>
<tr>
<td><code>in</code></td>
<td><code>jp</code> pointer to the job object to be posted</td>
</tr>
</tbody>
</table>
Function Class:
This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:

![Call Graph](call_graph.png)

### 7.44.4.7 chJobPostS()

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

Posts a job object.

**Note**

By design the object can be always immediately posted.

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>in</code></td>
<td><code>jqp</code></td>
</tr>
<tr>
<td><code>in</code></td>
<td><code>jp</code></td>
</tr>
</tbody>
</table>
7.44 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:

```
chJobPostS  chMBPostTimeoutS
chMBGetFreeCountI  chSchRescheduleS
chThdDequeueNextI
chThdEnqueueTimeoutS
```

### 7.44.4.8 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>in</td>
<td>jqp</td>
</tr>
<tr>
<td>in</td>
<td>jp</td>
</tr>
</tbody>
</table>
Function Class:
Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

![Call Graph](image)

### 7.44.4.9 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><code>jqp</code></td>
</tr>
<tr>
<td>in</td>
<td><code>jp</code></td>
</tr>
</tbody>
</table>
Function Class:

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

Here is the call graph for this function:

![Call Graph](call_graph.png)

### 7.44.4.10 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>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>in</strong></td>
<td>jqp</td>
</tr>
<tr>
<td><strong>in</strong></td>
<td>jp</td>
</tr>
</tbody>
</table>
Function Class:

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

Here is the call graph for this function:

```
7.44.4.11 chJobPostAhead()

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><strong>jqp</strong></td>
</tr>
<tr>
<td>in</td>
<td><strong>jp</strong></td>
</tr>
</tbody>
</table>
```
Function Class:
Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

```
7.44.4.12  chJobDispatch()

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

Waits for a job then executes it.

Parameters

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

Returns

- The function outcome.

Return values

<table>
<thead>
<tr>
<th>Return value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSG_OK</td>
<td>if a job has been executed.</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:

![Call Graph Diagram]

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

#### Returns

The function outcome.

#### Return values

<table>
<thead>
<tr>
<th>Return value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>MSG_OK</code></td>
<td>if a job has been executed.</td>
</tr>
<tr>
<td><code>MSG_TIMEOUT</code></td>
<td>if a timeout occurred.</td>
</tr>
<tr>
<td><code>MSG_RESET</code></td>
<td>if the internal mailbox has been reset.</td>
</tr>
<tr>
<td><code>MSG_JOB_NULL</code></td>
<td>if a <code>JOB_NULL</code> has been received.</td>
</tr>
</tbody>
</table>
Here is the call graph for this function:
7.45 Memory Management

7.45.1 Detailed Description

Memory Management services.

Modules

- Core Memory Manager
- Memory Heaps
- Memory Pools
7.46 Core Memory Manager

7.46.1 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.

7.46.2 Macro Definition Documentation

7.46.2.1 CH_CFG_MEMCORE_SIZE

#define CH_CFG_MEMCORE_SIZE 0

Managed RAM size.

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

Note

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

Requires CH_CFG_USE_MEMCORE.
7.46 Core Memory Manager

7.46.2.2 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.

7.46.2.3 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.

7.46.3 Typedef Documentation

7.46.3.1 memgetfunc_t

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

Memory get function.

7.46.3.2 memgetfunc2_t

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

Enhanced memory get function.
7.46.4 Function Documentation

7.46.4.1 __core_init()

```c
void __core_init ( void )
```

Low level memory manager initialization.

Function Class:

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

7.46.4.2 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

<table>
<thead>
<tr>
<th>Return</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>NULL</code></td>
<td>allocation failed, core memory exhausted.</td>
</tr>
</tbody>
</table>

Function Class:

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

Here is the call graph for this function:

```
chCoreAllocFromBaseI  chDbgCheckClassI   chSysHalt
```

7.46.4.3 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 \texttt{offset + size} bytes. The returned pointer has \texttt{offset} bytes before its address and \texttt{size} bytes after.

Parameters

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

Returns

A pointer to the allocated memory block.

Return values

```
\texttt{NULL} | allocation failed, core memory exhausted.
```

ChibiOS/RT
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:

```
chCoreAllocFromTopI  chDbgCheckClassI  chSysHalt
```

7.46.4.4 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>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:
Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

```
chCoreAllocFromBase -> chDbgCheckClassI
     |                    __stats_start_measure
     |                           _crit_thd
     |                        __dbg_check_lock
     |                     chSysLock
     |                  __dbg_check_unlock
     |                chSysUnlock
     |              __stats_stop_measure
     |         chCoreAllocFromBase
     |                  chSysLock
     |              __dbg_check_unlock
     |          __stats_start_measure
     |                 _crit_thd
     |                  chCoreAllocFromBase
```

### 7.46.4.5 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:

```
chCoreAllocFromTop
chCoreAllocFromTopI
chSysLock
chSysUnlock
chDbgCheckClassI
__dbg_check_lock
__stats_start_measure
__crit_thd
__dbg_check_unlock
__stats_stop_measure
__crit_thd
```

### 7.46.4.6 chCoreGetStatusX()

```c
size_t chCoreGetStatusX (void)
```

Core memory status.

Returns

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

Function Class:

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

### 7.46.4.7 chCoreAllocAlignedI()

```c
static void* chCoreAllocAlignedI (size_t size,
                                  unsigned align) [inline], [static]
```

Allocates a memory block.

The allocated block is guaranteed to be properly aligned to the specified alignment.

Note

This is a generic form with unspecified allocation position.
Parameters

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

Returns

A pointer to the allocated memory block.

Return values

- NULL: allocation failed, core memory exhausted.

Function Class:

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

### 7.46.4.8 chCoreAllocAligned()

```c
static void* chCoreAllocAligned ( size_t size, unsigned align ) [inline], [static]
```

Allocates a memory block.

The allocated block is guaranteed to be properly aligned to the specified alignment.

Note

This is a generic form with unspecified allocation position.

Parameters

| in | size | the size of the block to be allocated |
| in | align | desired memory alignment |

Returns

A pointer to the allocated memory block.

Return values

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

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

### 7.46.4.9 chCoreAllocI()

```c
static void* chCoreAllocI ( size_t size ) [inline], [static]
```

Allocates a memory block.

The allocated block is guaranteed to be properly aligned for a pointer data type.

Note

This is a generic form with unspecified allocation position.

Parameters

| In  | size | the size of the block to be allocated. |

Returns

A pointer to the allocated memory block.

Return values

- **NULL**: allocation failed, core memory exhausted.

Function Class:

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

### 7.46.4.10 chCoreAlloc()

```c
static void* chCoreAlloc ( size_t size ) [inline], [static]
```

Allocates a memory block.

The allocated block is guaranteed to be properly aligned for a pointer data type.

Note

This is a generic form with unspecified allocation position.
Parameters

| in  | size | the size of the block to be allocated. |

Returns

A pointer to the allocated memory block.

Return values

| NULL | allocation failed, core memory exhausted. |

Function Class:

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

7.46.5 Variable Documentation

7.46.5.1 ch_memcore

memcore_t ch_memcore

Memory core descriptor.
7.47 Memory Heaps

7.47.1 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.

7.47.2 Macro Definition Documentation

7.47.2.1 CH_HEAP_ALIGNMENT

#define CH_HEAP_ALIGNMENT 8U

Minimum alignment used for heap.

Note

Cannot use the sizeof operator in this macro.

7.47.2.2 CH_HEAP_AREA

#define CH_HEAP_AREA(
    name,
    size )

Value:

    ALIGNED_VAR(CH_HEAP_ALIGNMENT)
    uint8_t name[MEM_ALIGN_NEXT((size), CH_HEAP_ALIGNMENT)]

Allocation of an aligned static heap buffer.
7.47.3 Typedef Documentation

7.47.3.1 memory_heap_t

typedef struct memory_heap memory_heap_t

Type of a memory heap.

7.47.3.2 heap_header_t

typedef union heap_header heap_header_t

Type of a memory heap header.

7.47.4 Function Documentation

7.47.4.1 __heap_init()

void __heap_init ( void )

Initializes the default heap.

Function Class:

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

7.47.4.2 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 means that the effective heap size can be less than size.
Parameters

<table>
<thead>
<tr>
<th>out</th>
<th>heap</th>
<th>pointer to the memory heap descriptor to be initialized</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>buf</td>
<td>heap buffer base</td>
</tr>
<tr>
<td>in</td>
<td>size</td>
<td>heap size</td>
</tr>
</tbody>
</table>

Function Class:

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

### 7.47.4.3 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>in</th>
<th>heapp</th>
<th>pointer to a heap descriptor or NULL in order to access the default heap.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>size</td>
<td>the size of the block to be allocated. Note that the allocated block may be a bit bigger than the requested size for alignment and fragmentation reasons.</td>
</tr>
<tr>
<td>in</td>
<td>align</td>
<td>desired memory alignment</td>
</tr>
</tbody>
</table>

Returns

A pointer to the aligned allocated block.

Return values

- **NULL** if the block cannot be allocated.

Function Class:

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

### 7.47.4.4 chHeapFree()

```c
void chHeapFree ( void * p )
```

Frees a previously allocated memory block.
Parameters

| in  | p       | pointer to the memory block to be freed |

Function Class:

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

### 7.47.4.5 chHeapStatus()

```c
size_t chHeapStatus (  
    memory_heap_t * heapp,  
    size_t * totalp,  
    size_t * largestp )
```

Reports the heap status.

**Note**

This function is meant to be used in the test suite, it should not be really useful for the application code.

Parameters

| in  | heapp   | pointer to a heap descriptor or NULL in order to access the default heap. |
| in  | totalp  | pointer to a variable that will receive the total fragmented free space or NULL |
| in  | largestp | pointer to a variable that will receive the largest free free block found space or NULL |

Returns

The number of fragments in the heap.

Function Class:

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

### 7.47.4.6 chHeapAlloc()

```c
static void* chHeapAlloc (  
    memory_heap_t * heapp,  
    size_t size ) [inline], [static]
```

Allocates a block of memory from the heap by using the first-fit algorithm.

The allocated block is guaranteed to be properly aligned for a pointer data type.
Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>heap</th>
<th>pointer to a heap descriptor or NULL in order to access the default heap.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>size</td>
<td>the size of the block to be allocated. Note that the allocated block may be a bit bigger than the requested size for alignment and fragmentation reasons.</td>
</tr>
</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:

![Call Graph](image)

### 7.47.4.7 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

| in  | p      | pointer to the memory block |

ChibiOS/RT
Returns
   Size of the block.

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

7.47.5 Variable Documentation

7.47.5.1 default_heap

memory_heap_t default_heap [static]

Default heap descriptor.
7.48 Memory Pools

7.48.1 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 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 chGuardedPoolGetCounter (guarded_memory_pool_t *gmp)
  
  Gets the count of objects in a guarded memory pool.

- static void * chGuardedPoolAlloc (guarded_memory_pool_t *gmp)
  
  Allocates an object from a guarded memory pool.

- static void chGuardedPoolFree (guarded_memory_pool_t *gmp, void *objp)
  
  Releases an object into a guarded memory pool.

- static void chGuardedPoolAdd (guarded_memory_pool_t *gmp, void *objp)
  
  Adds an object to a guarded memory pool.

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

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

7.48.2 Macro Definition Documentation
7.48.2.1 __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>

7.48.2.2 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>

7.48.2.3 __GUARDEDMEMORYPOOL_DATA

#define __GUARDEDMEMORYPOOL_DATA(
    name,
    size,
    align )

ChibiOS/RT
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 <code>name</code></th>
<th>the name of the memory pool variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>in <code>size</code></td>
<td>size of the memory pool contained objects</td>
</tr>
<tr>
<td>in <code>align</code></td>
<td>required memory alignment</td>
</tr>
</tbody>
</table>

### 7.48.2.4 GUARDEDMEMORYPOOL_DECL

```c
#define GUARDEDMEMORYPOOL_DECL(
    name,
    size,
    align
)
guarded_memory_pool_t name = __GUARDEDMEMORYPOOL_DATA(name, size, align)
```

Static guarded memory pool initializer.

Statically initialized guarded memory pools require no explicit initialization using `chGuardedPoolInit()`.

**Parameters**

<table>
<thead>
<tr>
<th>in <code>name</code></th>
<th>the name of the guarded memory pool variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>in <code>size</code></td>
<td>size of the memory pool contained objects</td>
</tr>
<tr>
<td>in <code>align</code></td>
<td>required memory alignment</td>
</tr>
</tbody>
</table>

### 7.48.3 Function Documentation

#### 7.48.3.1 chPoolObjectInitAligned()

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

Initializes an empty memory pool.
7.48 Memory Pools

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>out mp</td>
<td>pointer to a memory_pool_t structure</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>
</tr>
<tr>
<td>in align</td>
<td>required memory alignment</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>
</tr>
</tbody>
</table>

Function Class:

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

7.48.3.2 chPoolLoadArray()

```c
void chPoolLoadArray ( 
    memory_pool_t * mp, 
    void * p, 
    size_t n )
```

Loads a memory pool with an array of static objects.

Precondition

The memory pool must already be initialized.

The array elements must be of the right size for the specified memory pool.

The array elements size must be a multiple of the alignment requirement for the pool.

Postcondition

The memory pool contains the elements of the input array.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in mp</td>
<td>pointer to a memory_pool_t structure</td>
</tr>
<tr>
<td>in p</td>
<td>pointer to the array first element</td>
</tr>
<tr>
<td>in n</td>
<td>number of elements in the array</td>
</tr>
</tbody>
</table>
Function Class:
Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

![Call Graph](image)

### 7.48.3.3 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:

7.48.3.4 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:

7.48.3.5 chPoolFreeI()

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>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>mp</td>
</tr>
<tr>
<td>in</td>
<td>objp</td>
</tr>
</tbody>
</table>
7.48 Memory Pools

Function 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](chPoolFree chDbgCheckClass chSysHalt)

7.48.3.6 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>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td><code>mp</code> pointer to a <code>memory_pool_t</code> structure</td>
</tr>
<tr>
<td>in</td>
<td><code>objp</code> the pointer to the object to be released</td>
</tr>
</tbody>
</table>
Function Class:

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

Here is the call graph for this function:

```
7.48.3.7 chGuardedPoolObjectInitAligned()

void chGuardedPoolObjectInitAligned (  
guarded_memory_pool_t * gmp,  
size_t size,  
unsigned align )

Initializes an empty guarded memory pool.

Parameters

<table>
<thead>
<tr>
<th>out gmp</th>
<th>pointer to a guarded_memory_pool_t structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>in size</td>
<td>the size of the objects contained in this guarded memory pool, the minimum accepted size is the size of a pointer to void.</td>
</tr>
<tr>
<td>in align</td>
<td>required memory alignment</td>
</tr>
</tbody>
</table>
```
Function Class:

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

Here is the call graph for this function:

```
chGuardedPoolObjectInitAligned
  ↓            ↓
chPoolObjectInitAligned  chSemObjectInit
                      ↓
ch_queue_init
```

7.48.3.8 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>
Function Class:

Normal API, 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
```

### 7.48.3.9 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 <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:

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 Diagram]

### 7.48.3.10 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 times out, the following special values are allowed:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <code>TIME_IMMEDIATE</code> immediate timeout.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <code>TIME_INFINITE</code> no timeout.</td>
</tr>
</tbody>
</table>

**Returns**

The pointer to the allocated object.
Return values

**NULL** if the operation timed out.

Function Class:

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

Here is the call graph for this function:

![Call Graph](image)

### 7.48.3.11 `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**

- **in** `gmp` pointer to a `guarded_memory_pool_t` structure
- **in** `objp` the pointer to the object to be released
Function Class:

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

Here is the call graph for this function:

```
chGuardedPoolFree
chGuardedPoolFreeI
chSchRescheduleS
chSysLock
chSysUnlock
chPoolFreeI
chSemSignalI
__sch_reschedule_ahead
chDbgCheckClassS
__dbg_check_lock
__stats_start_measure
_crit_thd
__dbg_check_unlock
__stats_stop_measure
_crit_thd
```

7.48.3.12 chPoolObjectInit()

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

Initializes an empty memory pool.

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>mp</td>
</tr>
<tr>
<td>in</td>
<td>size</td>
</tr>
<tr>
<td>in</td>
<td>provider</td>
</tr>
</tbody>
</table>
Function Class:

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

Here is the call graph for this function:

```
chPoolObjectInit → chPoolObjectInitAligned
```

### 7.48.3.13 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>in</th>
<th>mp</th>
<th>pointer to a <code>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>
Function Class:

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

Here is the call graph for this function:

```
chPoolAdd  chPoolFree   chPoolFreeI
            |      |           |
            |      |           |
chPoolAdd  chPoolFree   chSysLock
            |      |           |
            |      |           |
chPoolAdd  chPoolFreeI chSysUnlock
```

7.48.3.14  chPoolAddI()

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

Adds an object to a memory pool.

Precondition

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

Note

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

Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td><code>mp</code> pointer to a <code>memory_pool_t</code> structure</td>
</tr>
<tr>
<td>in</td>
<td><code>objp</code> 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 for chPoolAddI, chPoolFreeI, chDbgCheckClassI]

### 7.48.3.15 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 <code>guarded_memory_pool_t</code> structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>size</td>
<td>the size of the objects contained in this 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:

![Call Graph for chGuardedPoolObjectInit, chGuardedPoolObjectInitAligned, chPoolObjectInitAligned, chSemObjectInit]

### 7.48.3.16 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

\[ \text{in} \quad gmp \quad \text{pointer to a guarded_memory_pool_t structure} \]

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:

![Call Graph]

7.48.3.17 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

\[ \text{in} \quad gmp \quad \text{pointer to a guarded_memory_pool_t structure} \]

Returns

The pointer to the allocated object.
Return values

`NULL` if the pool is empty.

Function Class:

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

Here is the call graph for this function:

```
7.4.8.3.18 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

<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><code>objp</code></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.
7.48.3.19  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>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>gmp</td>
</tr>
<tr>
<td>in</td>
<td>objp</td>
</tr>
</tbody>
</table>
Function Class:

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

Here is the call graph for this function:

![Call Graph](image)

### 7.48.3.20 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>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>gmp</code></td>
<td>pointer to a <code>guarded_memory_pool_t</code> structure</td>
</tr>
<tr>
<td><code>objp</code></td>
<td>the pointer to the object to be added</td>
</tr>
</tbody>
</table>
7.48 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:

![Call Graph]

7.48.3.21 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 <code>guarded_memory_pool_t</code> 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)

### 7.48.3.22 chGuardedPoolAddS()

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

Adds an object to a guarded memory pool.

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

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

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td><code>gmp</code> pointer to a <code>guarded_memory_pool_t</code> structure</td>
</tr>
<tr>
<td>in</td>
<td><code>objp</code> the pointer to the object to be added</td>
</tr>
</tbody>
</table>
Function Class:

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

Here is the call graph for this function:
7.49 Complex Services

7.49.1 Detailed Description

Modules

- Objects FIFOs
- Objects Caches
- Dynamic Objects Factory
7.50 Objects FIFOs

7.50.1 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.
7.50.2 Typedef Documentation

7.50.2.1 objects_fifo_t

typedef struct ch_objects_fifo objects_fifo_t

Type of an objects FIFO.

7.50.3 Function Documentation

7.50.3.1 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>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 objalign</td>
<td>required objects alignment</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 with objalign alignment</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:

![Call Graph]

7.50.3.2 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:

![Call Graph]

### 7.50.3.3 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
```

### 7.50.3.4 chFifoTakeObjectTimeoutS()

```c
static void * chFifoTakeObjectTimeoutS ( objects_fifo_t * ofp, sysinterval_t timeout ) [inline], [static]
```

Allocates a free object.

**Parameters**

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

**Returns**

The pointer to the allocated object.

**Return values**

- **NULL** if an object is not available within the specified timeout.

Function Class:

This is an S-Class API, this function can be invoked from within a system lock zone by threads only.
Here is the call graph for this function:

```
chFifoTakeObjectTimeoutS
  └── chGuardedPoolAllocTimeoutS
      └── chPoolAllocS
          └── chSemWaitTimeoutS
```

### 7.50.3.5 chFifoTakeObjectTimeout()

```c
static void* chFifoTakeObjectTimeout ( 
    objects_fifo_t * ofp,
    sysinterval_t timeout ) [inline], [static]
```

Allocates a free object.

**Parameters**

<table>
<thead>
<tr>
<th>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:

7.50.3.6 chFifoReturnObjectI()

static void chFifoReturnObjectI (  
    objects_fifo_t * ofp,  
    void * objp ) [inline], [static]

Releases a fetched object.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>ofp</th>
<th>pointer to a objects_fifo_t structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>objp</td>
<td>pointer to the object to be released</td>
</tr>
</tbody>
</table>

Function Class:

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

Here is the call graph for this function:
7.50.3.7 chFifoReturnObjectS()

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](call_graph.png)

7.50.3.8 chFifoReturnObject()

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:

```
```

### 7.50.3.9  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

<table>
<thead>
<tr>
<th>in</th>
<th>ofp</th>
<th>pointer to a objects_fifo_t structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>objp</td>
<td>pointer to the object to be posted</td>
</tr>
</tbody>
</table>
Function Class:

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

Here is the call graph for this function:

![Call Graph Diagram]

### 7.50.3.10  `chFifoSendObjectS()`

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

Posts an object.

**Note**

By design the object can be always immediately posted.

**Parameters**

<table>
<thead>
<tr>
<th>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:

![Call Graph](image)

### 7.50.3.11 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:

```
chFifoSendObject chMBPostTimeout
chMBPostTimeoutS
chSysLock
chSysUnlock
```

### 7.50.3.12 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><code>ofp</code></td>
</tr>
<tr>
<td>in</td>
<td><code>objp</code></td>
</tr>
<tr>
<td>in</td>
<td><code>objp</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:

```
chFifoSendObjectAheadI chMBPostAheadI
   |                    chDbgCheckClassI
|                     chMBGetFreeCountI
|                          chThdDequeueNextI
```

### 7.50.3.13 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**

```
in    ofp    pointer to a objects_fifo_t structure
in    objp    pointer to the object to be posted
```

Function Class:

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

Here is the call graph for this function:

```
chFifoSendObjectAheadS chMBPostAheadTimeoutS
   |                    chDbgCheckClassS
|                     chMBGetFreeCountI
|                          chSchRescheduleS
|                                         chThdDequeueNextI
|                                                chThdEnqueueTimeoutS
```

ChibiOS/RT
7.50.3.14 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:

```
chFifoSendObjectAhead -> chMBPostAheadTimeoutI
                        ^
                        |  \       \       
                        |   \       \       
                        |    \       \       
                        |     \       \       
                        |      \       \       
```

7.50.3.15 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>

ChibiOS/RT
Returns

The operation status.

Return values

<table>
<thead>
<tr>
<th>MSG_OK</th>
<th>if an object has been correctly fetched.</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSG_TIMEOUT</td>
<td>if the FIFO is empty and a message cannot be fetched.</td>
</tr>
</tbody>
</table>

Function Class:

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

Here is the call graph for this function:

```
chFifoReceiveObjectI
   chMBFetchI
   chDbgCheckClassI
   chMBGetUsedCountI
   chThdDequeueNextI
```

7.50.3.16 chFifoReceiveObjectTimeoutS()

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

Fetches an object.

Parameters

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>in</strong></td>
<td><strong>ofp</strong></td>
</tr>
<tr>
<td><strong>in</strong></td>
<td><strong>objpp</strong></td>
</tr>
<tr>
<td><strong>in</strong></td>
<td><strong>timeout</strong></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Returns

The operation status.
Return values

<table>
<thead>
<tr>
<th>Message</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSG_OK</td>
<td>if an object has been correctly fetched.</td>
</tr>
<tr>
<td>MSG_TIMEOUT</td>
<td>if the operation has timed out.</td>
</tr>
</tbody>
</table>

Function Class:

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

Here is the call graph for this function:

```
chFifoReceiveObjectTimeoutS chMBFetchTimeoutS
chDbgCheckClassS chMBGetUsedCountI
chSchRescheduleS chThdDequeueNextI
chThdEnqueueTimeoutS
```

### 7.50.3.17 chFifoReceiveObjectTimeout()

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

Fetches an object.

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td><strong>ofp</strong> pointer to a <code>objects_fifo_t</code> structure</td>
</tr>
<tr>
<td>in</td>
<td><strong>objpp</strong> pointer to the fetched object reference</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>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:

![Call Graph Diagram](image-url)
7.51 Objects Caches

7.51.1 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.
7.51 Objects Caches

Functions

- static oc_object_t * hash_get_s (objects_cache_t *ocp, uint32_t group, uint32_t key)
  Returns an object pointer from the cache, if present.
- static oc_object_t * lru_get_last_s (objects_cache_t *ocp)
  Gets the least recently used object buffer from the LRU list.
- void chCacheObjectInit (objects_cache_t *ocp, ucnt_t hashn, oc_hash_header_t *hashp, ucnt_t objn, size_t objsz, void *objvp, oc_readf_t readf, oc_writef_t writef)
  Initializes a objects_cache_t object.
- oc_object_t * chCacheGetObject (objects_cache_t *ocp, uint32_t group, uint32_t key)
  Retrieves an object from the cache.
- void chCacheReleaseObjectI (objects_cache_t *ocp, oc_object_t *objp)
  Releases an object into the cache.
- bool chCacheReadObject (objects_cache_t *ocp, oc_object_t *objp, bool async)
  Reads object data from the storage.
- bool chCacheWriteObject (objects_cache_t *ocp, oc_object_t *objp, bool async)
  Writes the object data back to storage.
- static void chCacheReleaseObject (objects_cache_t *ocp, oc_object_t *objp)
  Releases an object into the cache.

7.51.2 Typedef Documentation

7.51.2.1 oc_flags_t

typedef uint32_t oc_flags_t

Flags of cached objects.

7.51.2.2 oc_hash_header_t

typedef struct ch_oc_hash_header oc_hash_header_t

Type of an hash element header.

7.51.2.3 oc_lru_header_t

typedef struct ch_oc_lru_header oc_lru_header_t

Type of an LRU element header.
7.51.2.4 oc_object_t

typedef struct ch_oc_object oc_object_t

Type of a cached object.

7.51.2.5 objects_cache_t

typedef struct ch_objects_cache objects_cache_t

Type of a cache object.

7.51.2.6 oc_readf_t

typedef bool(* oc_readf_t) (objects_cache_t *ocp, oc_object_t *objp, bool async)

Object read function.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in ocp</td>
<td>pointer to the objects_cache_t structure</td>
</tr>
<tr>
<td>in async</td>
<td>requests an asynchronous operation if supported, the function is then responsible for releasing the object</td>
</tr>
</tbody>
</table>

7.51.2.7 oc_writef_t

typedef bool(* oc_writef_t) (objects_cache_t *ocp, oc_object_t *objp, bool async)

Object write function.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in ocp</td>
<td>pointer to the objects_cache_t structure</td>
</tr>
<tr>
<td>in async</td>
<td>requests an asynchronous operation if supported, the function is then responsible for releasing the object</td>
</tr>
</tbody>
</table>

7.51.3 Function Documentation
### 7.51.3.1 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>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>ocp</td>
<td>pointer to the objects_cache_t structure to be</td>
</tr>
<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 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.

### 7.51.3.2 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>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>ocp</td>
<td>pointer to the objects_cache_t structure to be</td>
</tr>
</tbody>
</table>

**Returns**

The pointer to the retrieved object.

**Function Class:**

Not an API, this function is for internal use only.
Here is the call graph for this function:

![Call Graph Diagram]

### 7.51.3.3 chCacheObjectInit()

```c
void chCacheObjectInit ( 
    objects_cache_t *ocp, 
    ucnt_t hashn, 
    oc_hash_header_t *hashp, 
    ucnt_t objn, 
    size_t objsz, 
    void *objvp, 
    oc_readf_t readf, 
    oc_writef_t writef )
```

Initializes a `objects_cache_t` object.

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>out</strong></td>
<td><code>ocp</code> pointer to the <code>objects_cache_t</code> structure to be initialized</td>
</tr>
<tr>
<td><strong>in</strong></td>
<td><code>hashn</code> number of elements in the hash table array, must be a power of two and not lower than <code>objn</code></td>
</tr>
<tr>
<td><strong>in</strong></td>
<td><code>hashp</code> pointer to the hash table as an array of <code>oc_hash_header_t</code></td>
</tr>
<tr>
<td><strong>in</strong></td>
<td><code>objn</code> number of elements in the objects table array</td>
</tr>
<tr>
<td><strong>in</strong></td>
<td><code>objsz</code> size of elements in the objects table array, the minimum value is <code>sizeof (oc_object_t)</code></td>
</tr>
<tr>
<td><strong>in</strong></td>
<td><code>objvp</code> pointer to the hash objects as an array of structures starting with an <code>oc_object_t</code></td>
</tr>
<tr>
<td><strong>in</strong></td>
<td><code>readf</code> pointer to an object reader function</td>
</tr>
<tr>
<td><strong>in</strong></td>
<td><code>writef</code> pointer to an object writer function</td>
</tr>
</tbody>
</table>
Function Class:

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

Here is the call graph for this function:

```
+----------------+----------------+----------------+
| chCacheObjectInit | chSemObjectInit | ch_queue_init |
+----------------+----------------+----------------+
```

### 7.51.3.4 chCacheGetObject()

```c
oc_object_t * chCacheGetObject (oc_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><code>ocp</code></th>
<th>pointer to the <code>oc_objects_cache_t</code> structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td><code>group</code></td>
<td>object group identifier</td>
</tr>
<tr>
<td>in</td>
<td><code>key</code></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:

```
7.51.3.5  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

<table>
<thead>
<tr>
<th>in</th>
<th>ocp</th>
<th>pointer to the objects_cache_t structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>objp</td>
<td>pointer to the oc_object_t structure</td>
</tr>
</tbody>
</table>

Function Class:

This is an I-Class API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.
7.51.3.6 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

| in | ocp  | pointer to the objects_cache_t structure |
| in | objp | pointer to the oc_object_t structure     |
| in | async requests an asynchronous operation if supported, the function is then responsible for releasing the object |

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.

7.51.3.7 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>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in ocp</td>
<td>pointer to the objects_cache_t structure</td>
</tr>
<tr>
<td>in objp</td>
<td>pointer to the oc_object_t structure</td>
</tr>
<tr>
<td>in 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>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>false</td>
<td>if the operation succeeded.</td>
</tr>
<tr>
<td>true</td>
<td>if the synchronous write operation failed.</td>
</tr>
</tbody>
</table>

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

7.51.3.8 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:
7.52 Dynamic Objects Factory

7.52.1 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 \texttt{CH\_CFG\_USE\_MEMCORE} and \texttt{CH\_CFG\_USE\_MEMPOOLS} options to be set to \texttt{TRUE}. The option \texttt{CH\_CFG\_USE\_HEAP} is also required if the support for variable length objects is enabled.

Note

Compatible with RT and NIL.

Macros

- \#define \texttt{CH\_CFG\_FACTORY\_MAX\_NAMES\_LENGTH} 8
  
  Maximum length for object names.
- \#define \texttt{CH\_CFG\_FACTORY\_OBJECTS\_REGISTRY} \texttt{TRUE}
  
  Enables the registry of generic objects.
- \#define \texttt{CH\_CFG\_FACTORY\_GENERIC\_BUFFERS} \texttt{TRUE}
  
  Enables factory for generic buffers.
- \#define \texttt{CH\_CFG\_FACTORY\_SEMAPHORES} \texttt{TRUE}
  
  Enables factory for semaphores.
- \#define \texttt{CH\_CFG\_FACTORY\_SEMAPHORES} \texttt{FALSE}
  
  Enables factory for semaphores.
- \#define \texttt{CH\_CFG\_FACTORY\_MAILBOXES} \texttt{TRUE}
  
  Enables factory for mailboxes.
- \#define \texttt{CH\_CFG\_FACTORY\_MAILBOXES} \texttt{FALSE}
  
  Enables factory for mailboxes.
- \#define \texttt{CH\_CFG\_FACTORY\_OBJ\_FIFOS} \texttt{TRUE}
  
  Enables factory for objects FIFOs.
- \#define \texttt{CH\_CFG\_FACTORY\_OBJ\_FIFOS} \texttt{FALSE}
  
  Enables factory for objects FIFOs.
- \#define \texttt{CH\_CFG\_FACTORY\_PIPES} \texttt{TRUE}
  
  Enables factory for Pipes.
- \#define \texttt{CH\_CFG\_FACTORY\_PIPES} \texttt{FALSE}
  
  Enables factory for Pipes.
Typedefs

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

Data Structures

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

- `void __factory_init (void)`
  Initializes the objects factory.
- `registered_object_t * chFactoryRegisterObject (const char *name, void *objp)`
  Registers a generic object.
- `registered_object_t * chFactoryFindObject (const char *name)`
  Retrieves a registered object.
- `registered_object_t * chFactoryFindObjectByPointer (void *objp)`
  Retrieves a registered object by pointer.
- `void chFactoryReleaseObject (registered_object_t *rop)`
  Releases a registered object.
- `dyn_buffer_t * chFactoryCreateBuffer (const char *name, size_t size)`
  Creates a generic dynamic buffer object.
- `dyn_buffer_t * chFactoryFindBuffer (const char *name)`
  Retrieves a dynamic buffer object.
- `void chFactoryReleaseBuffer (dyn_buffer_t *dbp)`
  Releases a dynamic buffer object.
- `dyn_semaphore_t * chFactoryCreateSemaphore (const char *name, cnt_t n)`
  Creates a dynamic semaphore object.
- `dyn_semaphore_t * chFactoryFindSemaphore (const char *name)`
  Retrieves a dynamic semaphore object.
- `void chFactoryReleaseSemaphore (dyn_semaphore_t *dsp)`
  Releases a dynamic semaphore object.
- `dyn_mailbox_t * chFactoryCreateMailbox (const char *name, size_t n)`
  Creates a dynamic mailbox object.
- `dyn_mailbox_t * chFactoryFindMailbox (const char *name)`
  Retrieves a dynamic mailbox object.
- `void chFactoryReleaseMailbox (dyn_mailbox_t *dmp)`
  Releases a dynamic mailbox object.
- `dyn_objects_fifo_t * chFactoryCreateObjectsFIFO (const char *name, size_t objsize, size_t objn, unsigned objalign)`
  Creates a dynamic "objects FIFO" object.
- `dyn_objects_fifo_t * chFactoryFindObjectByPointer (void *objp)`
  Retrieves a dynamic "objects FIFO" object.
- `void chFactoryReleaseObjectsFIFO (dyn_objects_fifo_t *dofp)`
  Releases a dynamic "objects FIFO" object.
- `dyn_pipe_t * chFactoryCreatePipe (const char *name, size_t size)`
  Creates a dynamic pipe object.
- `dyn_pipe_t * 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.

7.52.2 Macro Definition Documentation

7.52.2.1 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.

7.52.2.2 CH_CFG_FACTORY_OBJECTS_REGISTRY

#define CH_CFG_FACTORY_OBJECTS_REGISTRY TRUE

Enables the registry of generic objects.

7.52.2.3 CH_CFG_FACTORY_GENERIC_BUFFERS

#define CH_CFG_FACTORY_GENERIC_BUFFERS TRUE

Enables factory for generic buffers.

7.52.2.4 CH_CFG_FACTORY_SEMAPHORES [1/2]

#define CH_CFG_FACTORY_SEMAPHORES TRUE

Enables factory for semaphores.
7.52.2.5 **CH_CFG_FACTORY_SEMAPHORES** [2/2]

```
#define CH_CFG_FACTORY_SEMAPHORES FALSE
```

Enables factory for semaphores.

7.52.2.6 **CH_CFG_FACTORY_MAILBOXES** [1/2]

```
#define CH_CFG_FACTORY_MAILBOXES TRUE
```

Enables factory for mailboxes.

7.52.2.7 **CH_CFG_FACTORY_MAILBOXES** [2/2]

```
#define CH_CFG_FACTORY_MAILBOXES FALSE
```

Enables factory for mailboxes.

7.52.2.8 **CH_CFG_FACTORY_OBJ_FIFOS** [1/3]

```
#define CH_CFG_FACTORY_OBJ_FIFOS TRUE
```

Enables factory for objects FIFOs.

7.52.2.9 **CH_CFG_FACTORY_OBJ_FIFOS** [2/3]

```
#define CH_CFG_FACTORY_OBJ_FIFOS TRUE
```

Enables factory for objects FIFOs.

7.52.2.10 **CH_CFG_FACTORY_OBJ_FIFOS** [3/3]

```
#define CH_CFG_FACTORY_OBJ_FIFOS FALSE
```

Enables factory for objects FIFOs.
7.52 Dynamic Objects Factory

7.52.2.11 CH_CFG_FACTORY_PIPES [1/2]

#define CH_CFG_FACTORY_PIPES TRUE

Enables factory for Pipes.

7.52.2.12 CH_CFG_FACTORY_PIPES [2/2]

#define CH_CFG_FACTORY_PIPES FALSE

Enables factory for Pipes.

7.52.3 Typedef Documentation

7.52.3.1 dyn_element_t

typedef struct ch_dyn_element dyn_element_t

Type of a dynamic object list element.

7.52.3.2 dyn_list_t

typedef struct chDynList dyn_list_t

Type of a dynamic object list.

7.52.3.3 registered_object_t

typedef struct ch_registered_static_object registered_object_t

Type of a registered object.

7.52.3.4 dyn_buffer_t

typedef struct ch_dyn_object dyn_buffer_t

Type of a dynamic buffer object.
7.52.3.5 dyn_semaphore_t

typedef struct ch_dyn_semaphore dyn_semaphore_t

Type of a dynamic semaphore.

7.52.3.6 dyn_mailbox_t

typedef struct ch_dyn_mailbox dyn_mailbox_t

Type of a dynamic buffer object.

7.52.3.7 dyn_objects_fifo_t

typedef struct ch_dyn_objects_fifo dyn_objects_fifo_t

Type of a dynamic buffer object.

7.52.3.8 dyn_pipe_t

typedef struct ch_dyn_pipe dyn_pipe_t

Type of a dynamic pipe object.

7.52.3.9 objects_factory_t

typedef struct ch_objects_factory objects_factory_t

Type of the factory main object.

7.52.4 Function Documentation
7.52.4.1 \texttt{\_\_factory\_init()}

\begin{verbatim}
void \_\_factory\_init( 
    void )
\end{verbatim}

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:

\begin{center}
\begin{tikzpicture}
  \node[align=center] (init) at (0,0) {	exttt{\_\_factory\_init}};
  \node[align=center] (mtx) at (1,1) {	exttt{chMtxObjectInit}};
  \node[align=center] (sem) at (1,-1) {	exttt{chSemObjectInit}};
  \node[align=center] (queue) at (1,0) {	exttt{ch\_queue\_init}};
  \draw[->] (init) -- (mtx);
  \draw[->] (init) -- (sem);
  \draw[->] (init) -- (queue);
\end{tikzpicture}
\end{center}

7.52.4.2 \texttt{chFactoryRegisterObject()}

\begin{verbatim}
registered\_object\_t * chFactoryRegisterObject( 
    const char * name, 
    void * objp )
\end{verbatim}

Registers a generic object.

Postcondition

A reference to the registered object is returned and the reference counter is initialized to one.

Parameters

\begin{tabular}{|c|l|}
\hline
\textbf{in} & \textit{name} & name to be assigned to the registered object \\
\textbf{in} & \textit{objp} & pointer to the object to be registered \\
\hline
\end{tabular}

Returns

The reference to the registered object.
Return values

<table>
<thead>
<tr>
<th>return value</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NULL</td>
<td>if the object to be registered cannot be allocated or a registered object with the same name exists.</td>
</tr>
</tbody>
</table>

Function Class:

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

### 7.52.4.3 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**

<table>
<thead>
<tr>
<th>type</th>
<th>name</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>name</td>
<td>name of the registered object</td>
</tr>
</tbody>
</table>

**Returns**

The reference to the found registered object.

**Return values**

<table>
<thead>
<tr>
<th>return value</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NULL</td>
<td>if a registered object with the specified name does not exist.</td>
</tr>
</tbody>
</table>

Function Class:

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

### 7.52.4.4 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.

7.52.4.5 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.

7.52.4.6 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

| in | name       | name to be assigned to the new dynamic buffer object |
|    | size       | payload size of the dynamic buffer object to be created |

Returns
The reference to the created dynamic buffer object.

Return values

| NULL | if the dynamic buffer object cannot be allocated or a dynamic buffer object with the same name exists. |

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

7.52.4.7 chFactoryFindBuffer()

`dyn_buffer_t * chFactoryFindBuffer (const char * name)`

Retrieves a dynamic buffer object.

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

Parameters

| in  | name       | name of the dynamic buffer object |

Returns
The reference to the found dynamic buffer object.

Return values

| NULL | if a dynamic buffer object with the specified name does not exist. |
7.52 Dynamic Objects Factory

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

7.52.4.8 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

<table>
<thead>
<tr>
<th>in</th>
<th>dbp</th>
<th>dynamic buffer 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.

7.52.4.9 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.

### 7.52.4.10 chFactoryFindSemaphore()

```c
dyn_semaphore_t * chFactoryFindSemaphore ( const char * name )
```

Retrieves a dynamic semaphore object.

Postcondition

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

Parameters

| in | name | name of the dynamic semaphore object |

Returns

The reference to the found dynamic semaphore object.

Return values

| NULL | if a dynamic semaphore object with the specified name does not exist. |

Function Class:

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

### 7.52.4.11 chFactoryReleaseSemaphore()

```c
void chFactoryReleaseSemaphore ( dyn_semaphore_t * dsp )
```

Releases a dynamic semaphore object.

The reference counter of the dynamic semaphore object is decreased by one, if reaches zero then the dynamic semaphore object memory is freed.
7.52 Dynamic Objects Factory

Parameters

| in  | dsp  | dynamic semaphore object reference |

Function Class:

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

7.52.4.12 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.

7.52.4.13 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.

7.52.4.14 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.

7.52.4.15 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

<table>
<thead>
<tr>
<th>in</th>
<th>name</th>
<th>name to be assigned to the new dynamic &quot;objects FIFO&quot; object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>objsize</td>
<td>size of objects</td>
</tr>
<tr>
<td>in</td>
<td>objn</td>
<td>number of objects available</td>
</tr>
<tr>
<td>in</td>
<td>objalign</td>
<td>required objects alignment</td>
</tr>
</tbody>
</table>

Returns

The reference to the created dynamic "objects FIFO" object.

Return values

| NULL | if the dynamic "objects FIFO" object cannot be allocated or a dynamic "objects FIFO" object with the same name exists. |

Function Class:

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

7.52.4.16 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.

### 7.52.4.17 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.

### 7.52.4.18 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.

7.52.4.19 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.

7.52.4.20 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

| in  | dpp | dynamic pipe object reference |

Function Class:

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

### 7.52.4.21 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

| in  | dep | pointer to the element field of the object |

Returns

The duplicated object reference.

Function Class:

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

### 7.52.4.22 chFactoryGetObject()

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

Returns the pointer to the inner registered object.

Parameters

| in  | rop | registered object reference |
Returns

The pointer to the registered object.

Function Class:

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

7.52.4.23 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:

![Call Graph](chFactoryGetBufferSize chHeapGetSize)

7.52.4.24 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.*

### 7.52.4.25 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.*

### 7.52.4.26 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.

7.52.4.27 chFactoryGetObjectsFIFO()

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

Returns the pointer to the inner objects FIFO.

Parameters

| in | dofpl | dynamic "objects FIFO" object reference |

Returns

The pointer to the objects FIFO.

Function Class:

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

7.52.4.28 chFactoryGetPipe()

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

Returns the pointer to the inner pipe.

Parameters

| in | dpl | dynamic pipe object reference |

Returns

The pointer to the pipe.

Function Class:

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

7.52.5 Variable Documentation

7.52.5.1 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 8

Data Structure Documentation

8.1 ch_binary_semaphore Struct Reference

Binary semaphore type.

#include <chbsem.h>

Inheritance diagram for ch_binary_semaphore:
Collaboration diagram for ch_binary_semaphore:

```
additional_inherited_members

8.1.1 Detailed Description

Binary semaphore type.

8.2 ch_delta_list Struct Reference

Delta list element and header structure.
#include <chlists.h>
Collaboration diagram for ch_delta_list:
```

ChibiOS/RT
8.2 ch_delta_list Struct Reference

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.

8.2.1 Detailed Description

Delta list element and header structure.

8.2.2 Field Documentation

8.2.2.1 next

`ch_delta_list_t* ch_delta_list::next`

Next in the delta list.

8.2.2.2 prev

`ch_delta_list_t* ch_delta_list::prev`

Previous in the delta list.

8.2.2.3 delta

`sysinterval_t ch_delta_list::delta`

Time interval from previous.
8.3 ch_dyn_element Struct Reference

Type of a dynamic object list element.

#include <chfactory.h>

Collaboration diagram for ch_dyn_element:

![Collaboration diagram](image)

**Data Fields**

- struct ch_dyn_element * next
  
  Next dynamic object in the list.

- ucnt_t refs
  
  Number of references to this object.

8.3.1 Detailed Description

Type of a dynamic object list element.

8.3.2 Field Documentation

8.3.2.1 next

struct ch_dyn_element* ch_dyn_element::next

Next dynamic object in the list.

8.3.2.2 refs

ucnt_t ch_dyn_element::refs

Number of references to this object.
8.4 ch_dyn_list Struct Reference

Type of a dynamic object list.

#include <chfactory.h>

Collaboration diagram for ch_dyn_list:

8.4.1 Detailed Description

Type of a dynamic object list.

8.5 ch_dyn_mailbox Struct Reference

Type of a dynamic buffer object.

#include <chfactory.h>
Collaboration diagram for ch_dyn_mailbox:

Data Fields

- **dyn_element_t element**
  
  List element of the dynamic buffer object.

- **mailbox_t mbx**
  
  The mailbox.

8.5.1 Detailed Description

Type of a dynamic buffer object.

8.5.2 Field Documentation
### 8.5.2.1 element

```
dyn_element_t ch_dyn_mailbox::element
```

List element of the dynamic buffer object.

### 8.5.2.2 mbx

```
mailbox_t ch_dyn_mailbox::mbx
```

The mailbox.

### 8.6 ch_dyn_object Struct Reference

Type of a dynamic buffer object.

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

Collaboration diagram for ch_dyn_object:

![Collaboration diagram for ch_dyn_object](image)

**Data Fields**

- `dyn_element_t element`

  List element of the dynamic buffer object.
8.6.1 Detailed Description

Type of a dynamic buffer object.

8.6.2 Field Documentation

8.6.2.1 element

dyn_element_t ch_dyn_object::element

List element of the dynamic buffer object.

8.7 ch_dyn_objects_fifo Struct Reference

Type of a dynamic buffer object.

#include <chfactory.h>

Collaboration diagram for ch_dyn_objects_fifo:
Data Fields

- `dyn_element_t element`
  List element of the dynamic buffer object.
- `objects_fifo_t fifo`
  The objects FIFO.

8.7.1 Detailed Description

Type of a dynamic buffer object.

8.7.2 Field Documentation

8.7.2.1 element

`dyn_element_t ch_dyn_objects_fifo::element`

List element of the dynamic buffer object.

8.7.2.2 fifo

`objects_fifo_t ch_dyn_objects_fifo::fifo`

The objects FIFO.

8.8 ch_dyn_pipe Struct Reference

Type of a dynamic pipe object.

#include <chfactory.h>
Data Structure Documentation

Collaboration diagram for ch_dyn_pipe:

Data Fields

- **dyn_element_t element**
  
  *List element of the dynamic pipe object.*

- **pipe_t pipe**
  
  *The pipe.*
### 8.8.1 Detailed Description

Type of a dynamic pipe object.

### 8.8.2 Field Documentation

#### 8.8.2.1 element

```c
dyn_element_t ch_dyn_pipe::element
```

List element of the dynamic pipe object.

#### 8.8.2.2 pipe

```c
pipe_t ch_dyn_pipe::pipe
```

The pipe.

### 8.9 ch_dyn_semaphore Struct Reference

Type of a dynamic semaphore.

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

Collaboration diagram for `ch_dyn_semaphore`:
Data Fields

- `dyn_element_t element`
  List element of the dynamic semaphore.
- `semaphore_t sem`
  The semaphore.

8.9.1 Detailed Description

Type of a dynamic semaphore.

8.9.2 Field Documentation

8.9.2.1 element

`dyn_element_t ch_dyn_semaphore::element`

List element of the dynamic semaphore.

8.9.2.2 sem

`semaphore_t ch_dyn_semaphore::sem`

The semaphore.

8.10 ch_job_descriptor Struct Reference

Type of a job descriptor.

#include <chjobs.h>

Collaboration diagram for ch_job_descriptor:
Data Fields

- `job_function_t jobfunc`
  
  Job function.

- `void * jobarg`
  
  Argument to be passed to the job function.

8.10.1 Detailed Description

Type of a job descriptor.

8.10.2 Field Documentation

8.10.2.1 jobfunc

```c
job_function_t ch_job_descriptor::jobfunc
```

Job function.

8.10.2.2 jobarg

```c
void* ch_job_descriptor::jobarg
```

Argument to be passed to the job function.

8.11 ch_jobs_queue Struct Reference

Type of a jobs queue.

```c
#include <chjobs.h>
```

ChibiOS/RT
Collaboration diagram for ch_jobs_queue:

Data Fields

- guarded_memory_pool_t free
  Pool of the free jobs.
- mailbox_t mbx
  Mailbox of the sent jobs.

8.11.1 Detailed Description

Type of a jobs queue.

8.11.2 Field Documentation
8.11.2.1 free

`guarded_memory_pool_t ch_jobs_queue::free`

Pool of the free jobs.

8.11.2.2 mbx

`mailbox_t ch_jobs_queue::mbx`

Mailbox of the sent jobs.

8.12 ch_list Struct Reference

Structure representing a generic single link list header and element.

```
#include <chlists.h>
```

Collaboration diagram for ch_list:

![Collaboration diagram for ch_list](image)

Data Fields

- `ch_list_t * next`
  
  Next in the list/queue.

8.12.1 Detailed Description

Structure representing a generic single link list header and element.

8.12.2 Field Documentation
8.12.2.1 next

\texttt{ch_list_t* ch_list::next}

Next in the list/queue.

8.13 \texttt{ch_mutex} Struct Reference

Mutex structure.

\texttt{#include <chmtx.h>}

Collaboration diagram for \texttt{ch_mutex}:

```
Data Fields

- \texttt{ch_queue_t queue}
  Queue of the threads sleeping on this mutex.

- \texttt{thread_t * owner}
  Owner \texttt{thread_t} pointer or NULL.

- \texttt{mutex_t * next}
  Next \texttt{mutex_t} into an owner-list or NULL.

- \texttt{cnt_t cnt}
  Mutex recursion counter.
```
8.13 ch_mutex Struct Reference

8.13.1 Detailed Description

Mutex structure.

8.13.2 Field Documentation

8.13.2.1 queue

ch_queue_t ch_mutex::queue

Queue of the threads sleeping on this mutex.

8.13.2.2 owner

thread_t* ch_mutex::owner

Owner thread_t pointer or NULL.

8.13.2.3 next

mutex_t* ch_mutex::next

Next mutex_t into an owner-list or NULL.

8.13.2.4 cnt

cnt_t ch_mutex::cnt

Mutex recursion counter.
8.14 ch_objects_cache Struct Reference

Structure representing a cache object.

#include <chobjcaches.h>

Collaboration diagram for ch_objects_cache:
Data Fields

- `ucnt_t hashn`
  Number of elements in the hash table.
- `oc_hash_header_t * hashp`
  Pointer to the hash table.
- `ucnt_t objn`
  Number of elements in the objects table.
- `size_t objsz`
  Size of elements in the objects table.
- `void * objvp`
  Pointer to the objects table.
- `oc_lru_header_t lru`
  LRU list header.
- `semaphore_t cache_sem`
  Semaphore for cache access.
- `semaphore_t lru_sem`
  Semaphore for LRU access.
- `oc_readf_t readf`
  Reader functions for cached objects.
- `oc_writef_t writef`
  Writer functions for cached objects.

8.14.1 Detailed Description

Structure representing a cache object.

8.14.2 Field Documentation

8.14.2.1 hashn

`ucnt_t ch_objects_cache::hashn`

Number of elements in the hash table.

8.14.2.2 hashp

`oc_hash_header_t * ch_objects_cache::hashp`

Pointer to the hash table.
8.14.2.3 objn

ucnt_t ch_objects_cache::objn

Number of elements in the objects table.

8.14.2.4 objsz

size_t ch_objects_cache::objsz

Size of elements in the objects table.

8.14.2.5 objvp

void* ch_objects_cache::objvp

Pointer to the objects table.

8.14.2.6 lru

oc_lru_header_t ch_objects_cache::lru

LRU list header.

8.14.2.7 cache_sem

semaphore_t ch_objects_cache::cache_sem

Semaphore for cache access.

8.14.2.8 lru_sem

semaphore_t ch_objects_cache::lru_sem

Semaphore for LRU access.
8.14.2.9 readf

c_readf_t ch_objects_cache::readf

Reader functions for cached objects.

8.14.2.10 writef

oc_writef_t ch_objects_cache::writef

Writer functions for cached objects.

8.15 ch_objects_factory Struct Reference

Type of the factory main object.

#include <chfactory.h>

Collaboration diagram for ch_objects_factory:
Data Structure Documentation

Data Fields

- **mutex_t mtx**
  
  Factory access mutex or semaphore.

- **dyn_list_t obj_list**
  
  List of the registered objects.

- **memory_pool_t obj_pool**
  
  Pool of the available registered objects.

- **dyn_list_t buf_list**
  
  List of the allocated buffer objects.

- **dyn_list_t sem_list**
  
  List of the allocated semaphores.

- **memory_pool_t sem_pool**
  
  Pool of the available semaphores.

- **dyn_list_t mbx_list**
  
  List of the allocated buffer objects.

- **dyn_list_t fifo_list**
  
  List of the allocated "objects FIFO" objects.

- **dyn_list_t pipe_list**
  
  List of the allocated pipe objects.

8.15.1 Detailed Description

Type of the factory main object.

8.15.2 Field Documentation

8.15.2.1 mtx

`mutex_t ch_objects_factory::mtx`

Factory access mutex or semaphore.

8.15.2.2 obj_list

`dyn_list_t ch_objects_factory::obj_list`

List of the registered objects.
8.15.2.3  obj_pool

memory_pool_t  ch_objects_factory::obj_pool

Pool of the available registered objects.

8.15.2.4  buf_list

dyn_list_t  ch_objects_factory::buf_list

List of the allocated buffer objects.

8.15.2.5  sem_list

dyn_list_t  ch_objects_factory::sem_list

List of the allocated semaphores.

8.15.2.6  sem_pool

memory_pool_t  ch_objects_factory::sem_pool

Pool of the available semaphores.

8.15.2.7  mbx_list

dyn_list_t  ch_objects_factory::mbx_list

List of the allocated buffer objects.

8.15.2.8  fifo_list

dyn_list_t  ch_objects_factory::fifo_list

List of the allocated "objects FIFO" objects.
8.15.2.9 pipe_list

dyn_list_t ch_objects_factory::pipe_list

List of the allocated pipe objects.

8.16 ch_objects_fifo Struct Reference

Type of an objects FIFO.

#include <chobjfifos.h>

Collaboration diagram for ch_objects_fifo:

Data Fields

- guarded_memory_pool_t free
  Pool of the free objects.
- mailbox_t mbx
  Mailbox of the sent objects.
8.16.1 Detailed Description

Type of an objects FIFO.

8.16.2 Field Documentation

8.16.2.1 free

`guarded_memory_pool_t ch_objects_fifo::free`

Pool of the free objects.

8.16.2.2 mbx

`mailbox_t ch_objects_fifo::mbx`

Mailbox of the sent objects.

8.17 ch_oc_hash_header Struct Reference

Structure representing an hash table element.

```
#include <chobjcaches.h>
```

ChibiOS/RT
Collaboration diagram for ch_oc_hash_header:

Data Fields

- oc_object_t * hash_next
  
  Next in the collisions list.
- oc_object_t * hash_prev
  
  Previous in the collisions list.

8.17.1 Detailed Description

Structure representing an hash table element.

8.17.2 Field Documentation
8.17.2.1 hash_next

\texttt{oc\_object\_t* ch\_oc\_hash\_header::hash\_next}

Next in the collisions list.

8.17.2.2 hash_prev

\texttt{oc\_object\_t* ch\_oc\_hash\_header::hash\_prev}

Previous in the collisions list.

8.18 ch\_oc\_lru\_header Struct Reference

Structure representing an hash table element.

\#include \texttt{<chobjcaches.h>}

Collaboration diagram for ch\_oc\_lru\_header:
Data Fields

- `oc_object_t * hash_next
  Next in the collisions list.
- `oc_object_t * hash_prev
  Previous in the collisions list.
- `oc_object_t * lru_next
  Next in the LRU list.
- `oc_object_t * lru_prev
  Previous in the LRU list.

8.18.1 Detailed Description

Structure representing an hash table element.

8.18.2 Field Documentation

8.18.2.1 hash_next

`oc_object_t* ch_oc_lru_header::hash_next

Next in the collisions list.

8.18.2.2 hash_prev

`oc_object_t* ch_oc_lru_header::hash_prev

Previous in the collisions list.

8.18.2.3 lru_next

`oc_object_t* ch_oc_lru_header::lru_next

Next in the LRU list.

8.18.2.4 lru_prev

`oc_object_t* ch_oc_lru_header::lru_prev

Previous in the LRU list.
8.19 ch_oc_object Struct Reference

Structure representing a cached object.

#include <chobjcaches.h>

Collaboration diagram for ch_oc_object:

Data Fields

- oc_object_t * hash_next
  Next in the collisions list.
- oc_object_t * hash_prev
  Previous in the collisions list.
- oc_object_t * lru_next
  Next in the LRU list.
- oc_object_t * lru_prev
  Previous in the LRU list.
- uint32_t obj_group
  Object group.
- uint32_t obj_key
  Object key.
- semaphore_t obj_sem
Semaphore for object access.

- `oc_flags_t obj_flags`
  Object flags.
- `void * dptr`
  User pointer.

### 8.19.1 Detailed Description

Structure representing a cached object.

### 8.19.2 Field Documentation

#### 8.19.2.1 `hash_next`

```c
oc_object_t* ch_oc_object::hash_next
```

Next in the collisions list.

#### 8.19.2.2 `hash_prev`

```c
oc_object_t* ch_oc_object::hash_prev
```

Previous in the collisions list.

#### 8.19.2.3 `lru_next`

```c
oc_object_t* ch_oc_object::lru_next
```

Next in the LRU list.

#### 8.19.2.4 `lru_prev`

```c
oc_object_t* ch_oc_object::lru_prev
```

Previous in the LRU list.
8.19.2.5 obj_group

uint32_t ch_oc_object::obj_group

Object group.

8.19.2.6 obj_key

uint32_t ch_oc_object::obj_key

Object key.

8.19.2.7 obj_sem

semaphore_t ch_oc_object::obj_sem

Semaphore for object access.

8.19.2.8 obj_flags

oc_flags_t ch_oc_object::obj_flags

Object flags.

8.19.2.9 dptr

void* ch_oc_object::dptr

User pointer.

Note

This pointer can be used to refer to external buffers, chCacheObjectInit() initializes it to NULL.
8.20  ch_os_instance Struct Reference

System instance data structure.

#include <chobjects.h>

Collaboration diagram for ch_os_instance:

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`
8.20.1 Detailed Description

System instance data structure.

8.20.2 Field Documentation

8.20.2.1 rlist

`ready_list_t ch_os_instance::rlist`

Ready list header.

8.20.2.2 vtlist

`virtual_timers_list_t ch_os_instance::vtlist`

Virtual timers delta list header.

8.20.2.3 reglist

`registry_t ch_os_instance::reglist`

Registry header.

Note

This field is present only if the SMP mode is disabled.

8.20.2.4 core_id

`core_id_t ch_os_instance::core_id`

Core associated to this instance.
8.20.2.5 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.

8.20.2.6 config

const os_instance_config_t* ch_os_instance::config
Pointer to the instance configuration data.

8.20.2.7 mainthread

thread_t ch_os_instance::mainthread
Main thread descriptor.

8.20.2.8 dbg

system_debug_t ch_os_instance::dbg
System debug.

8.20.2.9 trace_buffer

trace_buffer_t ch_os_instance::trace_buffer
Trace buffer.

8.20.2.10 kernel_stats

kernel_stats_t ch_os_instance::kernel_stats
Global kernel statistics.
8.21 ch_os_instance_config Struct Reference

Type of an system instance configuration.

#include <chobjects.h>

Collaboration diagram for ch_os_instance_config:

```
8.21.2 Field Documentation

8.21.2.1 name

const char* ch_os_instance_config::name

Instance name.
```
8.21.2.2 mainthread_base

```c
stkalign_t* ch_os_instance_config::mainthread_base
```

Lower limit of the main function thread stack.

8.21.2.3 mainthread_end

```c
stkalign_t* ch_os_instance_config::mainthread_end
```

Upper limit of the main function thread stack.

8.21.2.4 idlethread_base

```c
stkalign_t* ch_os_instance_config::idlethread_base
```

Lower limit of the dedicated idle thread stack.

8.21.2.5 idlethread_end

```c
stkalign_t* ch_os_instance_config::idlethread_end
```

Upper limit of the dedicated idle thread stack.

8.22 ch_priority_queue Struct Reference

Structure representing a generic priority-ordered bidirectional linked list header and element.

```c
#include <chlists.h>
```

Collaboration diagram for ch_priority_queue:
8.22 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.

8.22.1 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.

8.22.2 Field Documentation

8.22.2.1 next

`ch_priority_queue_t* ch_priority_queue::next

Next in the queue.

8.22.2.2 prev

`ch_priority_queue_t* ch_priority_queue::prev

Previous in the queue.

8.22.2.3 prio

`tprio_t ch_priority_queue::prio

Priority of this element.
8.23  ch_queue Struct Reference

Structure representing a generic bidirectional linked list header and element.

#include <chlists.h>

Collaboration diagram for ch_queue:

\begin{figure}
\centering
\includegraphics[width=0.3\textwidth]{ch_queue}
\caption{Collaboration diagram for ch_queue.}
\end{figure}

Data Fields

- ch_queue_t * next
  
  Next in the list/queue.

- ch_queue_t * prev
  
  Previous in the queue.

8.23.1  Detailed Description

Structure representing a generic bidirectional linked list header and element.

8.23.2  Field Documentation

8.23.2.1  next

ch_queue_t* ch_queue::next

Next in the list/queue.
8.23.2.2 prev

ch_queue_t* ch_queue::prev

Previous in the queue.

8.24 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.
8.24.1 Detailed Description

Type of a ready list header.

8.24.2 Field Documentation

8.24.2.1 pqueue

ch_priority_queue_t ch_ready_list::pqueue
Threads ordered queues header.

Note
The priority field must be initialized to zero.

8.24.2.2 current

thread_t* ch_ready_list::current
The currently running thread.

8.25 ch_registered_static_object Struct Reference

Type of a registered object.
#include <chfactory.h>

Collaboration diagram for ch_registered_static_object:
Data Fields

- **dyn_element_t element**
  
  List element of the registered object.

- **void * objp**
  
  Pointer to the object.

### 8.25.1 Detailed Description

Type of a registered object.

### 8.25.2 Field Documentation

#### 8.25.2.1 element

```
dyn_element_t  ch_registered_static_object::element
```

List element of the registered object.

#### 8.25.2.2 objp

```
void*  ch_registered_static_object::objp
```

Pointer to the object.

**Note**

The type of the object is not stored in anyway.
8.26 ch_registry Struct Reference

Type of a registry structure.

#include <chobjects.h>

Collaboration diagram for ch_registry:

![Collaboration diagram for ch_registry](image)

Data Fields

- ch_queue_t queue

  Registry queue header.

8.26.1 Detailed Description

Type of a registry structure.

8.26.2 Field Documentation

8.26.2.1 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.

#### 8.27.1 Detailed Description

Type of an RFCU structure.

#### 8.27.2 Field Documentation

##### 8.27.2.1 mask

`rfcu_mask_t ch_rfcu::mask`

Mask of the pending runtime faults.
8.28 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
8.28.1 Detailed Description

Semaphore structure.

8.28.2 Field Documentation

8.28.2.1 queue

ch_queue_t ch_semaphore::queue

Queue of the threads sleeping on this semaphore.

8.28.2.2 cnt

cnt_t ch_semaphore::cnt

The semaphore counter.

8.29 ch_system Struct Reference

Type of system data structure.

#include <chobjects.h>
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.

8.29.1 Detailed Description

Type of system data structure.

8.29.2 Field Documentation
8.29.1 state

system_state_t ch_system::state

Operating system state.

8.29.2 instances

os_instance_t* ch_system::instances[PORT_CORES_NUMBER]

Initialized OS instances or NULL.

8.29.3 tmc

tm_calibration_t ch_system::tmc

Time measurement calibration data.

8.29.4 reglist

registry_t ch_system::reglist

Registry header.

Note
This field is present only if the SMP mode is enabled.

8.29.5 rfcu

rfcu_t ch_system::rfcu

Runtime Faults Collection Unit.

Note
This field is present only if the SMP mode is enabled.
8.30 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.

8.30.1 Detailed Description

System debug data structure.

8.30.2 Field Documentation

8.30.2.1 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.
### 8.31 ch_thread Struct Reference

Structure representing a thread.

```c
#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.
8.31 ch_thread Struct Reference

- `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.

8.31.1 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.

8.31.2 Field Documentation

8.31.2.1 list

`ch_list_t ch_thread::list`

Threads lists element.

8.31.2.2 queue

`ch_queue_t ch_thread::queue`

Threads queues element.

8.31.2.3 pqueue

`ch_priority_queue_t ch_thread::pqueue`

Threads ordered queues element.
8.31.2.4 hdr

union { ... } ch_thread::hdr

Shared list headers.

8.31.2.5 ctx

struct port_context ch_thread::ctx

Processor context.

8.31.2.6 rqueue

ch_queue_t ch_thread::rqueue

Registry queue element.

8.31.2.7 owner

os_instance_t* ch_thread::owner

OS instance owner of this thread.

8.31.2.8 name

const char* ch_thread::name

Thread name or NULL.

8.31.2.9 wabase

stkalign_t* ch_thread::wabase

Working area base address.

Note

This pointer is used for stack overflow checks and for dynamic threading.
### 8.31.2.10 state

`tstate_t ch_thread::state`

Current thread state.

### 8.31.2.11 flags

`tmode_t ch_thread::flags`

Various thread flags.

### 8.31.2.12 refs

`trefs_t ch_thread::refs`

References to this thread.

### 8.31.2.13 ticks

`tslices_t ch_thread::ticks`

Number of ticks remaining to this thread.

### 8.31.2.14 time

`volatile systime_t ch_thread::time`

Thread consumed time in ticks.

**Note**

This field can overflow.
8.31.2.15 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.

8.31.2.16 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.

8.31.2.17 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.

8.31.2.18 wtrp

thread_reference_t* ch_thread::wtrp

Pointer to a generic thread reference object.

Note
This field is used to get a pointer to a synchronization object and is valid when the thread is in CH_STATE_SUSPENDED state.
8.31.2.19  sentmsg

msg_t ch_thread::sentmsg

Thread sent message.

8.31.2.20  wtsem

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.

8.31.2.21  wtmutex

struct ch_mutex* ch_thread::wtmutex

Pointer to a generic mutex object.

Note
This field is used to get a pointer to a synchronization object and is valid when the thread is in CH_STATE←_WTMTX state.

8.31.2.22  ewmask

eventmask_t ch_thread::ewmask

Enabled events mask.

Note
This field is only valid while the thread is in the CH_STATE_WTOREVT or CH_STATE_WTANDEVT states.
8.31.2.23  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.

8.31.2.24  waiting

ch_list_t ch_thread::waiting

Termination waiting list.

8.31.2.25  msgqueue

ch_queue_t ch_thread::msgqueue

Messages queue.

8.31.2.26  epending

eventmask_t ch_thread::epending

Pending events mask.

8.31.2.27  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.
8.31.2.28 realprio

tprio_t ch_thread::realprio

Thread’s own, non-inherited, priority.

8.31.2.29 mpool

void* ch_thread::mpool

Memory Pool where the thread workspace is returned.

8.31.2.30 stats

time_measurement_t ch_thread::stats

Thread statistics.

8.32 ch_threads_queue Struct Reference

Type of a threads queue.

#include <chobjects.h>

Collaboration diagram for ch_threads_queue:
Data Fields

- `ch_queue_t queue`
  Threads queue header.

### 8.32.1 Detailed Description

Type of a threads queue.

### 8.32.2 Field Documentation

#### 8.32.2.1 queue

```c
ch_queue_t ch_threads_queue::queue
```

Threads queue header.

### 8.33 ch_virtual_timer Struct Reference

Structure representing a Virtual Timer.

```c
#include <chobjects.h>
```

Collaboration diagram for ch_virtual_timer:
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.*

8.33.1 Detailed Description

Structure representing a Virtual Timer.

8.33.2 Field Documentation

8.33.2.1 dlist

`ch_delta_list_t ch_virtual_timer::dlist`

Delta list element.

8.33.2.2 func

`vtfunc_t ch_virtual_timer::func`

Timer callback function pointer.

8.33.2.3 par

`void * ch_virtual_timer::par`

Timer callback function parameter.

8.33.2.4 reload

`sysinterval_t ch_virtual_timer::reload`

Current reload interval.
8.34  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.

8.34.1 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.
8.34.2 Field Documentation

8.34.2.1 dlist

ch_delta_list_t ch_virtual_timers_list::dlist

Delta list header.

8.34.2.2 systime

volatile systime_t ch_virtual_timers_list::systime

System Time counter.

8.34.2.3 lasttime

systime_t ch_virtual_timers_list::lasttime

System time of the last tick event.

8.34.2.4 laststamp

volatile uint64_t ch_virtual_timers_list::laststamp

Last generated time stamp.
8.35  chdebug_t Struct Reference

ChibiOS/RT memory signature record.

#include <chregistry.h>

Collaboration diagram for chdebug_t:

```
chdebug_t
+ identifier
+ zero
+ size
+ version
+ ptrsize
+ timesize
+ threadsize
+ off_prio
+ off_ctx
+ off_newer
and 8 more...
```

Data Fields

- **char identifier [4]**
  
  *Always set to "main".*

- **uint8_t zero**

  *Must be zero.*

- **uint8_t size**

  *Size of this structure.*

- **uint16_t version**

  *Encoded ChibiOS/RT version.*

- **uint8_t ptrsize**

  *Size of a pointer.*

- **uint8_t timesize**

  *Size of a systime_t.*

- **uint8_t threadsize**

  *Size of a thread_t.*

- **uint8_t off_prio**

  *Offset of prio field.*

- **uint8_t off_ctx**
8.35.1 Detailed Description

ChibiOS/RT memory signature record.

8.35.2 Field Documentation

8.35.2.1 identifier

char chdebug_t::identifier[4]

Always set to "main".

8.35.2.2 zero

uint8_t chdebug_t::zero

Must be zero.
8.35.2.3 size

uint8_t chdebug_t::size

Size of this structure.

8.35.2.4 version

uint16_t chdebug_t::version

Encoded ChibiOS/RT version.

8.35.2.5 ptrsize

uint8_t chdebug_t::ptrsize

Size of a pointer.

8.35.2.6 timesize

uint8_t chdebug_t::timesize

Size of a systime_t.

8.35.2.7 threadsize

uint8_t chdebug_t::threadsize

Size of a thread_t.

8.35.2.8 off_prio

uint8_t chdebug_t::off_prio

Offset of prio field.
8.35.2.9  off_ctx

uint8_t chdebug_t::off_ctx

Offset of ctx field.

8.35.2.10  off_newer

uint8_t chdebug_t::off_newer

Offset of newer field.

8.35.2.11  off_older

uint8_t chdebug_t::off_older

Offset of older field.

8.35.2.12  off_name

uint8_t chdebug_t::off_name

Offset of name field.

8.35.2.13  off_stklimit

uint8_t chdebug_t::off_stklimit

Offset of stklimit field.

8.35.2.14  off_state

uint8_t chdebug_t::off_state

Offset of state field.
8.35.2.15  off_flags

uint8_t chdebug_t::off_flags
Offset of flags field.

8.35.2.16  off_refs

uint8_t chdebug_t::off_refs
Offset of refs field.

8.35.2.17  off_preempt

uint8_t chdebug_t::off_preempt
Offset of preempt field.

8.35.2.18  off_time

uint8_t chdebug_t::off_time
Offset of time field.

8.36  condition_variable Struct Reference

condition_variable_t structure.
#include <chcond.h>
Collaboration diagram for condition_variable:
8.37 event_listener Struct Reference

Event Listener structure.

#include <chevents.h>

Collaboration diagram for event_listener:
Data Fields

- `event_listener_t * next`
  Next Event Listener registered on the event source.

- `thread_t * listener`
  Thread interested in the event source.

- `eventmask_t events`
  Events to be set in the listening thread.

- `eventflags_t flags`
  Flags added to the listener by the event source.

- `eventflags_t wflags`
  Flags that this listener interested in.

8.37.1 Detailed Description

Event Listener structure.

8.37.2 Field Documentation

8.37.2.1 `next`

`event_listener_t * event_listener::next`

Next Event Listener registered on the event source.

8.37.2.2 `listener`

`thread_t * event_listener::listener`

Thread interested in the event source.

8.37.2.3 `events`

`eventmask_t event_listener::events`

Events to be set in the listening thread.
8.37.4 flags

```c
eventflags_t event_listener::flags
```

Flags added to the listener by the event source.

8.37.5 wflags

```c
eventflags_t event_listener::wflags
```

Flags that this listener interested in.

---

8.38 event_source Struct Reference

Event Source structure.

```c
#include <chevents.h>
```

ChibiOS/RT
Collaboration diagram for event_source:

Data Fields

- `event_listener_t * next`

  *First Event Listener registered on the Event Source.*

8.38.1 Detailed Description

Event Source structure.

8.38.2 Field Documentation
8.38.2.1 next

```
event_listener_t* event_source::next
```

First Event Listener registered on the Event Source.

8.39 guarded_memory_pool_t Struct Reference

Guarded memory pool descriptor.

```
#include <chmempools.h>
```

Collaboration diagram for guarded_memory_pool_t:

![Collaboration diagram for guarded_memory_pool_t](image)

### Data Fields

- **semaphore_t sem**
  
  Counter semaphore guarding the memory pool.

- **memory_pool_t pool**
  
  The memory pool itself.
8.39.1 Detailed Description

Guarded memory pool descriptor.

8.39.2 Field Documentation

8.39.2.1 sem

`semaphore_t guarded_memory_pool_t::sem`

Counter semaphore guarding the memory pool.

8.39.2.2 pool

`memory_pool_t guarded_memory_pool_t::pool`

The memory pool itself.

8.40 heap_header Union Reference

Memory heap block header.

#include <chmemheaps.h>

Collaboration diagram for heap_header:
8.40.1 Detailed Description

Memory heap block header.

8.40.2 Field Documentation

8.40.2.1 next

heap_header_t* heap_header::next

Next block in free list.

8.40.2.2 pages

size_t heap_header::pages

Size of the area in pages.

8.40.2.3 heap

memory_heap_t* heap_header::heap

Block owner heap.

8.40.2.4 size

size_t heap_header::size

Size of the area in bytes.
8.41 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.

8.41.1 Detailed Description

Type of a kernel statistics structure.
8.41.2 Field Documentation

8.41.2.1 n_irq

`ucnt_t kernel_stats_t::n_irq`

Number of IRQs.

8.41.2.2 n_ctxswc

`ucnt_t kernel_stats_t::n_ctxswc`

Number of context switches.

8.41.2.3 m_crit_thd

`time_measurement_t kernel_stats_t::m_crit_thd`

Measurement of threads critical zones duration.

8.41.2.4 m_crit_isr

`time_measurement_t kernel_stats_t::m_crit_isr`

Measurement of ISRs critical zones duration.
8.42 mailbox_t Struct Reference

Structure representing a mailbox object.

```c
#include <chmboxes.h>
```

Collaboration diagram for mailbox_t:

![Collaboration Diagram](image)

**Data Fields**

- `msg_t * buffer`
  
  Pointer to the mailbox buffer.

- `msg_t * top`
  
  Pointer to the location after the buffer.

- `msg_t * wrptr`
  
  Write pointer.

- `msg_t * rdptr`
  
  Read pointer.
8.42 mailbox_t Struct Reference

- **size_t cnt**
  
  Messages in queue.

- **bool reset**
  
  True in reset state.

- **threads_queue_t qw**
  
  Queued writers.

- **threads_queue_t qr**
  
  Queued readers.

### 8.42.1 Detailed Description

Structure representing a mailbox object.

### 8.42.2 Field Documentation

#### 8.42.2.1 buffer

`msg_t* mailbox_t::buffer`

Pointer to the mailbox buffer.

#### 8.42.2.2 top

`msg_t* mailbox_t::top`

Pointer to the location after the buffer.

#### 8.42.2.3 wrptr

`msg_t* mailbox_t::wrptr`

Write pointer.
8.42.2.4 rdptr

```c
msg_t* mailbox_t::rdptr
```

Read pointer.

8.42.2.5 cnt

```c
size_t mailbox_t::cnt
```

Messages in queue.

8.42.2.6 reset

```c
bool mailbox_t::reset
```

True in reset state.

8.42.2.7 qw

```c
threads_queue_t mailbox_t::qw
```

Queued writers.

8.42.2.8 qr

```c
threads_queue_t mailbox_t::qr
```

Queued readers.
8.43 memcore_t Struct Reference

Type of memory core object.

#include <chmemcore.h>

Collaboration diagram for memcore_t:

Data Fields

- uint8_t* basemem
  
  Next free address.
- uint8_t* topmem
  
  Final address.

8.43.1 Detailed Description

Type of memory core object.

8.43.2 Field Documentation

8.43.2.1 basemem

uint8_t* memcore_t::basemem

Next free address.

8.43.2.2 topmem

uint8_t* memcore_t::topmem

Final address.
8.44 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.

8.44.1 Detailed Description

Structure describing a memory heap.

8.44.2 Field Documentation

8.44.2.1 provider

`memgetfunc2_t memory_heap::provider`

Memory blocks provider for this heap.

8.44.2.2 header

`heap_header_t memory_heap::header`

Free blocks list header.

8.44.2.3 mtx

`mutex_t memory_heap::mtx`

Heap access mutex.
8.45 memory_pool_t Struct Reference

Memory pool descriptor.

#include <chmempools.h>

Collaboration diagram for memory_pool_t:

![Collaboration diagram](image)

### Data Fields

- struct pool_header * next
  
  Pointer to the header.

- size_t object_size
  
  Memory pool objects size.

- unsigned align
  
  Required alignment.

- memgetfunc_t provider
  
  Memory blocks provider for this pool.

8.45.1 Detailed Description

Memory pool descriptor.

8.45.2 Field Documentation
8.45.2.1 next

struct pool_header* memory_pool_t::next

Pointer to the header.

8.45.2.2 object_size

size_t memory_pool_t::object_size

Memory pool objects size.

8.45.2.3 align

unsigned memory_pool_t::align

Required alignment.

8.45.2.4 provider

memgetfunc_t memory_pool_t::provider

Memory blocks provider for this pool.

8.46 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`
8.46 pipe_t Struct Reference

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.

8.46.1 Detailed Description

Structure representing a pipe object.

8.46.2 Field Documentation

8.46.2.1 buffer

uint8_t* pipe_t::buffer

Pointer to the pipe buffer.

8.46.2.2 top

uint8_t* pipe_t::top

Pointer to the location after the buffer.
8.46.2.3 wrptr

uint8_t* pipe_t::wrptr

Write pointer.

8.46.2.4 rdptr

uint8_t* pipe_t::rdptr

Read pointer.

8.46.2.5 cnt

size_t pipe_t::cnt

Bytes in the pipe.

8.46.2.6 reset

bool pipe_t::reset

True if in reset state.

8.46.2.7 wtr

thread_reference_t pipe_t::wtr

Waiting writer.

8.46.2.8 rtr

thread_reference_t pipe_t::rtr

Waiting reader.
8.46.2.9  cmtx

`mutex_t pipe_t::cmtx`

Common access mutex.

8.46.2.10  wmtx

`mutex_t pipe_t::wmtx`

Write access mutex.

8.46.2.11  rmtx

`mutex_t pipe_t::rmtx`

Read access mutex.

8.47  pool_header Struct Reference

Memory pool free object header.

```
#include <chmempools.h>
```

Collaboration diagram for pool_header:

```
+next
          ▶
          ◀
          ▶
          ◀
```

Data Fields

- `struct pool_header * next`

  Pointer to the next pool header in the list.
8.47.1 Detailed Description

Memory pool free object header.

8.47.2 Field Documentation

8.47.2.1 next

struct pool_header* pool_header::next

Pointer to the next pool header in the list.

8.48 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.

- `tfunc_t funcp`
  
  Thread function pointer.

- `void * arg`
  
  Thread argument.

- `os_instance_t * instance`
  
  OS instance affinity or `NULL` for current one.

### 8.48.1 Detailed Description

Type of a thread descriptor.

### 8.48.2 Field Documentation

#### 8.48.2.1 name

```c
const char* thread_descriptor_t::name
```

Thread name.

#### 8.48.2.2 wbase

```c
stkalign_t* thread_descriptor_t::wbase
```

Pointer to the working area base.

#### 8.48.2.3 wend

```c
stkalign_t* thread_descriptor_t::wend
```

Pointer to the working area end.
8.48.2.4 prio

tprio_t thread_descriptor_t::prio

Thread priority.

8.48.2.5 funcp

tfunc_t thread_descriptor_t::funcp

Thread function pointer.

8.48.2.6 arg

void* thread_descriptor_t::arg

Thread argument.

8.48.2.7 instance

os_instance_t* thread_descriptor_t::instance

OS instance affinity or NULL for current one.

8.49 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.

### 8.49.1 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.

### 8.49.2 Field Documentation

#### 8.49.2.1 best

```
rtcnt_t time_measurement_t::best
```

Best measurement.

#### 8.49.2.2 worst

```
rtcnt_t time_measurement_t::worst
```

Worst measurement.
8.49.2.3 last

rtcnt_t time_measurement_t::last

Last measurement.

8.49.2.4 n

ucnt_t time_measurement_t::n

Number of measurements.

8.49.2.5 cumulative

rttime_t time_measurement_t::cumulative

Cumulative measurement.

8.50 tm_calibration_t Struct Reference

Type of a time measurement calibration data.

#include <chtm.h>

Collaboration diagram for tm_calibration_t:

Data Fields

- rtcnt_t offset
  
  Measurement calibration value.
8.50.1 Detailed Description

Type of a time measurement calibration data.

8.50.2 Field Documentation

8.50.2.1 offset

**rtcnt_t** tm_calibration_t::offset

Measurement calibration value.

8.51 trace_buffer_t Struct Reference

Trace buffer header.

```
#include <chtrace.h>
```

ChibiOS/RT
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.
• trace_event_t buffer [CH_DBG_TRACE_BUFFER_SIZE]
  Ring buffer.

8.51.1 Detailed Description

Trace buffer header.

8.51.2 Field Documentation

8.51.2.1 suspended

uint16_t trace_buffer_t::suspended

Suspended trace sources mask.

8.51.2.2 size

uint16_t trace_buffer_t::size

Trace buffer size (entries).

8.51.2.3 ptr

trace_event_t* trace_buffer_t::ptr

Pointer to the buffer front.

8.51.2.4 buffer

trace_event_t trace_buffer_t::buffer[CH_DBG_TRACE_BUFFER_SIZE]

Ring buffer.
8.52 trace_event_t Struct Reference

Trace buffer record.

#include <chtrace.h>

Collaboration diagram for trace_event_t:

![Collaboration Diagram]

Data Fields

- uint32_t type:3
  Record type.
- uint32_t state:5
  Switched out thread state.
- uint32_t rstamp: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.
8.52 trace_event_t Struct Reference

• struct {
  thread_t * ntp
  \textit{Switched in thread.}
  void * wtobjp
  \textit{Object where going to sleep.}
} sw

  \textit{Structure representing a context switch.}

• thread_t * tp
  \textit{Thread made ready.}

• msg_t msg
  \textit{Ready message.}

• struct {
  thread_t * tp
  \textit{Thread made ready.}
  msg_t msg
  \textit{Ready message.}
} rdy

  \textit{Structure representing a thread becoming ready.}

• const char * name
  \textit{ISR function name taken using \texttt{func}.}

• struct {
  const char * name
  \textit{ISR function name taken using \texttt{func}.}
} isr

  \textit{Structure representing an ISR enter.}

• const char * reason
  \textit{Halt error string.}

• struct {
  const char * reason
  \textit{Halt error string.}
} halt

  \textit{Structure representing an halt.}

• void * up1
  \textit{Trace user parameter 1.}

• void * up2
  \textit{Trace user parameter 2.}

• struct {
  void * up1
  \textit{Trace user parameter 1.}
  void * up2
  \textit{Trace user parameter 2.}
} user

  \textit{User trace structure.}

\section{8.52.1 Detailed Description}

Trace buffer record.
8.52.2 Field Documentation

8.52.2.1 type

\texttt{uint32_t trace_event_t::type}

Record type.

8.52.2.2 state

\texttt{uint32_t trace_event_t::state}

Switched out thread state.

8.52.2.3 rtstamp

\texttt{uint32_t trace_event_t::rtstamp}

Accurate time stamp.

\textbf{Note}

This field only available if the post supports \texttt{PORT_SUPPORTS_RT} else it is set to zero.

8.52.2.4 time

\texttt{systime_t trace_event_t::time}

System time stamp of the switch event.

8.52.2.5 ntp

\texttt{thread_t* trace_event_t::ntp}

Switched in thread.
8.52.2.6 **wtobjp**

```c
void* trace_event_t::wtobjp
```

Object where going to sleep.

8.52.2.7 **sw**

```c
struct { ... } trace_event_t::sw
```

Structure representing a context switch.

8.52.2.8 **tp**

```c
thread_t* trace_event_t::tp
```

Thread made ready.

8.52.2.9 **msg**

```c
msg_t trace_event_t::msg
```

Ready message.

8.52.2.10 **rdy**

```c
struct { ... } trace_event_t::rdy
```

Structure representing a thread becoming ready.

8.52.2.11 **name**

```c
const char* trace_event_t::name
```

ISR function name taken using `func`.
8.52.2.12 isr

struct { ... } trace_event_t::isr

Structure representing an ISR enter.

8.52.2.13 reason

const char* trace_event_t::reason

Halt error string.

8.52.2.14 halt

struct { ... } trace_event_t::halt

Structure representing an halt.

8.52.2.15 up1

void* trace_event_t::up1

Trace user parameter 1.

8.52.2.16 up2

void* trace_event_t::up2

Trace user parameter 2.

8.52.2.17 user

struct { ... } trace_event_t::user

User trace structure.
Chapter 9

File Documentation

9.1 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 "chalignt.h"
#include "chtrace.h"
#include "chport.h"
#include "chtm.h"
#include "chstats.h"
#include "chobjects.h"
#include "chsyst.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 0
  Stable release flag.

ChibiOS/RT version identification

• #define CH_KERNEL_VERSION "7.0.0"
  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 0
  Kernel version patch number.

Constants for configuration options

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

9.1.1 Detailed Description

ChibiOS/RT main include file.

9.2 chalign.h File Reference

Memory alignment macros and structures.

Macros

Memory alignment support macros

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

9.2.1 Detailed Description

Memory alignment macros and structures.
9.3  chbsem.h File Reference

Binary semaphores structures and macros.

Data Structures

• struct ch_binary_semaphore
  Binary semaphore type.

Macros

• #define __BSEMAPHORE_DATA(name, taken) {__SEMAPHORE_DATA(name.sem, ((taken) ? 0 : 1))}
  Data part of a static semaphore initializer.
• #define BSEMAPHORE_DECL(name, taken) binary_semaphore_t name = __BSEMAPHORE_DATA(name, taken)
  Static semaphore initializer.

Typedefs

• typedef struct ch_binary_semaphore binary_semaphore_t
  Binary semaphore type.

Functions

• static void chBSemObjectInit (binary_semaphore_t ∗bsp, bool taken)
  Initializes a binary semaphore.
• static msg_t chBSemWait (binary_semaphore_t ∗bsp)
  Wait operation on the binary semaphore.
• static msg_t chBSemWaitS (binary_semaphore_t ∗bsp)
  Wait operation on the binary semaphore.
• static msg_t chBSemWaitTimeoutS (binary_semaphore_t ∗bsp, sysinterval_t timeout)
  Wait operation on the binary semaphore.
• static msg_t chBSemWaitTimeout (binary_semaphore_t ∗bsp, sysinterval_t timeout)
  Wait operation on the binary semaphore.
• static void chBSemResetI (binary_semaphore_t ∗bsp, bool taken)
  Reset operation on the binary semaphore.
• static void chBSemReset (binary_semaphore_t ∗bsp, bool taken)
  Reset operation on the binary semaphore.
• static void chBSemSignalI (binary_semaphore_t ∗bsp)
  Performs a signal operation on a binary semaphore.
• static void chBSemSignal (binary_semaphore_t ∗bsp)
  Performs a signal operation on a binary semaphore.
• static bool chBSemGetStateI (const binary_semaphore_t ∗bsp)
  Returns the binary semaphore current state.
9.3.1 Detailed Description

Binary semaphores structures and macros.

Binary semaphores related APIs and services.

Operation mode

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

- **Taken**, when its counter has a value of zero or lower than zero. A negative number 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.

9.4 chchecks.h File Reference

Configuration file checks header.

9.4.1 Detailed Description

Configuration file checks header.

9.5 chcond.c File Reference

Condition Variables code.

#include "ch.h"
Functions

- `void chCondObjectInit (condition_variable_t *cp)`
  Initializes a `condition_variable_t` structure.
- `void chCondSignal (condition_variable_t *cp)`
  Signals one thread that is waiting on the condition variable.
- `void chCondSignalAll (condition_variable_t *cp)`
  Signals one thread that is waiting on the condition variable.
- `void chCondBroadcast (condition_variable_t *cp)`
  Signals all threads that are waiting on the condition variable.
- `void chCondBroadcastI (condition_variable_t *cp)`
  Signals all threads that are waiting on the condition variable.
- `msg_t chCondWait (condition_variable_t *cp)`
  Waits on the condition variable releasing the mutex lock.
- `msg_t chCondWaitS (condition_variable_t *cp)`
  Waits on the condition variable releasing the mutex lock.
- `msg_t chCondWaitTimeout (condition_variable_t *cp, sysinterval_t timeout)`
  Waits on the condition variable releasing the mutex lock.
- `msg_t chCondWaitTimeoutS (condition_variable_t *cp, sysinterval_t timeout)`
  Waits on the condition variable releasing the mutex lock.

9.5.1 Detailed Description

Condition Variables code.

9.6 chcond.h File Reference

Condition Variables macros and structures.

Data Structures

- `struct condition_variable`
  `condition_variable_t` structure.

Macros

- `#define __CONDVAR_DATA(name) {__CH_QUEUE_DATA(name.queue)}`
  Data part of a static condition variable initializer.
- `#define 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 the 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.

9.6.1 Detailed Description

Condition Variables macros and structures.

9.7 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.

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.
9.7.1 Detailed Description

Configuration file template.

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

9.8 chcustomer.h File Reference

Customer-related info.

Macros

- #define CH_CUSTOMER_ID_STRING "Santa, North Pole"  
  Customer readable identifier.
- #define CH_CUSTOMER_ID_CODE "xxxx-yyyy"  
  Customer code.
- #define CH_CUSTOMER_LICENSE_EOS_DATE 209912  
  End-Of-Support date (yyyymm).
- #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 (yyymmm).

### 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`

### 9.8.1 Detailed Description

Customer-related info.

### 9.9 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().
• void __dbg_check_enter_isr (void)
  Guard code for CH_IRQ_PROLOGUE().
• void __dbg_check_leave_isr (void)
  Guard code for CH_IRQ_EPILOGUE().
• void chDbgCheckClassI (void)
  I-class functions context check.
• void chDbgCheckClassS (void)
  S-class functions context check.

9.9.1 Detailed Description

Debug support code.

9.10 chdebug.h File Reference

Debug support macros and structures.

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.
9.10.1 Detailed Description

Debug support macros and structures.

9.11 chdelegates.c File Reference

Delegate threads code.

#include "ch.h"

Functions

• msg_t __ch_delegate_fn0 (va_list *argsp)
  Veneer for functions with no parameters.
• msg_t __ch_delegate_fn1 (va_list *argsp)
  Veneer for functions with one parameter.
• msg_t __ch_delegate_fn2 (va_list *argsp)
  Veneer for functions with two parameters.
• msg_t __ch_delegate_fn3 (va_list *argsp)
  Veneer for functions with three parameters.
• msg_t __ch_delegate_fn4 (va_list *argsp)
  Veneer for functions with four parameters.
• msg_t chDelegateCallVeneer (thread_t *tp, delegate_veneer_t veneer,...)
  Triggers a function call on a delegate thread.
• void chDelegateDispatch (void)
  Call messages dispatching.
• msg_t chDelegateDispatchTimeout (sysinterval_t timeout)
  Call messages dispatching with timeout.

9.11.1 Detailed Description

Delegate threads code.

Delegate threads.

Operation mode

A delegate thread is a thread performing function calls triggered by other threads. This functionality is especially useful when encapsulating a library not designed for threading into a delegate thread. Other threads have access to the library without having to worry about mutual exclusion.

Precondition

In order to use the pipes APIs the CH_CFG_USE_DELEGATES option must be enabled in chconf.h.

Note

Compatible with RT and NIL.
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.
9.12.1 Detailed Description

Delegate threads macros and structures.

9.13 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.

9.13.1 Detailed Description

Dynamic threads code.

9.14 chdynamic.h File Reference

Dynamic threads macros and structures.

Functions

- `thread_t * chThdCreateFromHeap (memory_heap_t *heapp, size_t size, const char *name, tprio_t prio, tfunc_t pf, void *arg)`
  - Creates a new thread allocating the memory from the heap.

- `thread_t * chThdCreateFromMemoryPool (memory_pool_t *mp, const char *name, tprio_t prio, tfunc_t pf, void *arg)`
  - Creates a new thread allocating the memory from the specified memory pool.

9.14.1 Detailed Description

Dynamic threads macros and structures.
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.

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.
9.15.1 Detailed Description

Early forward types declarations header.

9.16 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.
9.16.1 Detailed Description

Events code.

9.17 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)
C clears the pending events specified in the events mask.

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

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

- eventflags_t chEvtGetAndClearFlagsI (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.

9.17.1 Detailed Description

Events macros and structures.
ChibiOS objects factory and registry code.

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

### Functions

- `void __factory_init (void)`
  *Initializes the objects factory.*

- `registered_object_t * chFactoryRegisterObject (const char *name, void *objp)`
  *Registers a generic object.*

- `registered_object_t * chFactoryFindObject (const char *name)`
  *Retrieves a registered object.*

- `registered_object_t * chFactoryFindObjectByPointer (void *objp)`
  *Retrieves a registered object by pointer.*

- `void chFactoryReleaseObject (registered_object_t *rop)`
  *Releases a registered object.*

- `dyn_buffer_t * chFactoryCreateBuffer (const char *name, size_t size)`
  *Creates a generic dynamic buffer object.*

- `dyn_buffer_t * chFactoryFindBuffer (const char *name)`
  *Retrieves a dynamic buffer object.*

- `void chFactoryReleaseBuffer (dyn_buffer_t *dbp)`
  *Releases a dynamic buffer object.*

- `dyn_semaphore_t * chFactoryCreateSemaphore (const char *name, cnt_t n)`
  *Creates a dynamic semaphore object.*

- `dyn_semaphore_t * chFactoryFindSemaphore (const char *name)`
  *Retrieves a dynamic semaphore object.*

- `void chFactoryReleaseSemaphore (dyn_semaphore_t *dsp)`
  *Releases a dynamic semaphore object.*

- `dyn_mailbox_t * chFactoryCreateMailbox (const char *name, size_t n)`
  *Creates a dynamic mailbox object.*

- `dyn_mailbox_t * chFactoryFindMailbox (const char *name)`
  *Retrieves a dynamic mailbox object.*

- `void chFactoryReleaseMailbox (dyn_mailbox_t *dmp)`
  *Releases a dynamic mailbox object.*

- `dyn_objects_fifo_t * chFactoryCreateObjectsFIFO (const char *name, size_t objsize, size_t objn, unsigned objalign)`
  *Creates a dynamic "objects FIFO" object.*

- `dyn_objects_fifo_t * chFactoryFindObjectByPointer (void *objp)`
  *Retrieves a dynamic "objects FIFO" object.*

- `void chFactoryReleaseObjectsFIFO (dyn_objects_fifo_t *dofp)`
  *Releases a dynamic "objects FIFO" object.*

- `dyn_pipe_t * chFactoryCreatePipe (const char *name, size_t size)`
  *Creates a dynamic pipe object.*

- `dyn_pipe_t * chFactoryFindObjectByPointer (void *objp)`
  *Retrieves a dynamic pipe object.*

- `void chFactoryReleasePipe (dyn_pipe_t *dpp)`
  *Releases a dynamic pipe object.*
Variables

• objects_factory_t ch_factory
  Factory object static instance.

9.18.1 Detailed Description

ChibiOS objects factory and registry code.

9.19 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 Pipes.
• #define CH_CFG_FACTORY_PIPES TRUE

Enables factory for semaphores.
• #define CH_CFG_FACTORY_SEMAPHORES FALSE

Enables factory for mailboxes.
• #define CH_CFG_FACTORY_MAILBOXES FALSE

Enables factory for Pipes.
• #define CH_CFG_FACTORY_PIPES FALSE

Enabled factory for objects FIFOs.
• #define CH_CFG_FACTORY_OBJ_FIFOS FALSE

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.

• `dyn_semaphore_t * chFactoryFindSemaphore (const char *name)`
  Retrieves a dynamic semaphore object.

• `void chFactoryReleaseSemaphore (dyn_semaphore_t *dsp)`
  Releases a dynamic semaphore object.

• `dyn_mailbox_t * chFactoryCreateMailbox (const char *name, size_t n)`
  Creates a dynamic mailbox object.

• `dyn_mailbox_t * chFactoryFindMailbox (const char *name)`
  Retrieves a dynamic mailbox object.

• `void chFactoryReleaseMailbox (dyn_mailbox_t *dmp)`
  Releases a dynamic mailbox object.

• `dyn_objects_fifo_t * chFactoryCreateObjectsFIFO (const char *name, size_t objsize, size_t objn, unsigned objalign)`
  Creates a dynamic "objects FIFO" object.

• `dyn_objects_fifo_t * chFactoryFindObjectsFIFO (const char *name)`
  Retrieves a dynamic "objects FIFO" object.

• `void chFactoryReleaseObjectsFIFO (dyn_objects_fifo_t *dofp)`
  Releases a dynamic "objects FIFO" object.

• `dyn_pipe_t * chFactoryCreatePipe (const char *name, size_t size)`
  Creates a dynamic pipe object.

• `dyn_pipe_t * chFactoryFindPipe (const char *name)`
  Retrieves a dynamic pipe object.

• `void chFactoryReleasePipe (dyn_pipe_t *dpp)`
  Releases a dynamic pipe object.

• `static dyn_element_t * chFactoryDuplicateReference (dyn_element_t *dep)`
  Duplicates an object reference.

• `static void * chFactoryGetObject (registered_object_t *rop)`
  Returns the pointer to the inner registered object.

• `static size_t chFactoryGetBufferSize (dyn_buffer_t *dbp)`
  Returns the size of a generic dynamic buffer object.

• `static uint8_t * chFactoryGetBuffer (dyn_buffer_t *dbp)`
  Returns the pointer to the inner buffer.

• `static semaphore_t * chFactoryGetSemaphore (dyn_semaphore_t *dsp)`
  Returns the pointer to the inner semaphore.

• `static mailbox_t * chFactoryGetMailbox (dyn_mailbox_t *dmp)`
  Returns the pointer to the inner mailbox.

• `static objects_fifo_t * chFactoryGetObjectsFIFO (dyn_objects_fifo_t *dofp)`
  Returns the pointer to the inner objects FIFO.

• `static pipe_t * chFactoryGetPipe (dyn_pipe_t *dpp)`
  Returns the pointer to the inner pipe.

### 9.19.1 Detailed Description

ChibiOS objects factory structures and macros.
9.20  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.

9.20.1  Detailed Description

OS instances code.

9.21  chinstances.h File Reference

OS instances macros and structures.

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

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

9.21.1  Detailed Description

OS instances macros and structures.

9.22  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.
9.22.1 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.

9.23 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 0`
  Stable release flag.

ChibiOS/LIB version identification

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

Functions

- `static void __oslib_init (void)`
  Initialization of all library modules.
9.23.1 Detailed Description

ChibiOS/LIB main include file.

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

9.24 \texttt{chlicense.h} File Reference

License Module macros and structures.

\begin{verbatim}
#include "chversion.h"
#include "chcustomer.h"
\end{verbatim}

Macros

\begin{itemize}
  \item \#define \texttt{CH_LICENSE_TYPE_STRING} "GNU General Public License 3 (GPL3)"
    \textit{License identification string.}
  \item \#define \texttt{CH_LICENSE_ID_STRING} "N/A"
    \textit{Customer identification string.}
  \item \#define \texttt{CH_LICENSE_ID_CODE} "N/A"
    \textit{Customer code.}
  \item \#define \texttt{CH_LICENSE_MODIFIABLE_CODE} TRUE
    \textit{Code modifiability restrictions.}
  \item \#define \texttt{CH_LICENSE_FEATURES} CH\_FEATURES\_FULL
    \textit{Code functionality restrictions.}
  \item \#define \texttt{CH_LICENSE_MAX_DEPLOY} CH\_DEPLOY\_UNLIMITED
    \textit{Code deploy restrictions.}
\end{itemize}

Allowed Features Levels

\begin{itemize}
  \item \#define \texttt{CH\_FEATURES\_BASIC} 0
  \item \#define \texttt{CH\_FEATURES\_INTERMEDIATE} 1
  \item \#define \texttt{CH\_FEATURES\_FULL} 2
\end{itemize}

Deployment Options

\begin{itemize}
  \item \#define \texttt{CH\_DEPLOY\_UNLIMITED} -1
  \item \#define \texttt{CH\_DEPLOY\_NONE} 0
\end{itemize}

Licensing Options

\begin{itemize}
  \item \#define \texttt{CH\_LICENSE\_GPL} 0
  \item \#define \texttt{CH\_LICENSE\_GPL\_EXCEPTION} 1
  \item \#define \texttt{CH\_LICENSE\_COMMERCIAL\_FREE} 2
  \item \#define \texttt{CH\_LICENSE\_COMMERCIAL\_DEV\_1000} 3
  \item \#define \texttt{CH\_LICENSE\_COMMERCIAL\_DEV\_5000} 4
  \item \#define \texttt{CH\_LICENSE\_COMMERCIAL\_FULL} 5
  \item \#define \texttt{CH\_LICENSE\_COMMERCIAL\_RUNTIME} 6
  \item \#define \texttt{CH\_LICENSE\_PARTNER} 7
\end{itemize}
9.24.1 Detailed Description

License Module macros and structures.

9.25 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 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 *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.
9.25.1 Detailed Description

Lists and Queues header.

9.26 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.

9.26.1 Detailed Description

Mailboxes code.

9.27 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 chMBObjectInit (mailbox_t *mbp, msg_t *buf, size_t n)
  
  Initializes a mailbox object.

- void chMBReset (mailbox_t *mbp)
  
  Resets a mailbox object.

- void chMBResetI (mailbox_t *mbp)
  
  Resets a mailbox object.

- msg_t chMBPostTimeout (mailbox_t *mbp, msg_t msg, sysinterval_t timeout)
  
  Posts a message into a mailbox.

- msg_t chMBPostTimeoutS (mailbox_t *mbp, msg_t msg, sysinterval_t timeout)
  
  Posts a message into a mailbox.

- msg_t chMBPostI (mailbox_t *mbp, msg_t msg)
  
  Posts a message into a mailbox.

- msg_t chMBPostAheadTimeout (mailbox_t *mbp, msg_t msg, sysinterval_t timeout)
  
  Posts an high priority message into a mailbox.

- msg_t chMBPostAheadTimeoutS (mailbox_t *mbp, msg_t msg, sysinterval_t timeout)
  
  Posts an high priority message into a mailbox.

- msg_t chMBPostAheadI (mailbox_t *mbp, msg_t msg)
  
  Posts an high priority message into a mailbox.

- msg_t chMBFetchTimeout (mailbox_t *mbp, msg_t *msgp, sysinterval_t timeout)
  
  Retrieves a message from a mailbox.

- msg_t chMBFetchTimeoutS (mailbox_t *mbp, msg_t *msgp, sysinterval_t timeout)
  
  Retrieves a message from a mailbox.

- msg_t chMBFetchI (mailbox_t *mbp, msg_t *msgp)
  
  Retrieves a message from a mailbox.

- static size_t chMBGetSizeI (const mailbox_t *mbp)
  
  Returns the mailbox buffer size as number of messages.

- static size_t chMBGetUsedCountI (const mailbox_t *mbp)
  
  Returns the number of used message slots into a mailbox.

- static size_t chMBGetFreeCountI (const mailbox_t *mbp)
  
  Returns the number of free message slots into a mailbox.

- static msg_t chMBPeekI (const mailbox_t *mbp)
  
  Returns the next message in the queue without removing it.

- static void chMBResumeX (mailbox_t *mbp)
  
  Terminates the reset state.
9.27.1 Detailed Description

Mailboxes macros and structures.

9.28 chmemcore.c File Reference

Core memory manager code.

#include "ch.h"

Functions

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

Variables

- memcore_t ch_memcore
  Memory core descriptor.

9.28.1 Detailed Description

Core memory manager code.

9.29 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 chCoreAllocFromTop
  Allocates a memory block.
- #define chCoreAllocAlignedWithOffsetI chCoreAllocFromTopI
  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 ∗chCoreAlloc2 (size_t size)
  Allocates a memory block.

9.29.1 Detailed Description

Core memory manager macros and structures.

9.30 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.

9.30.1 Detailed Description

Memory heaps code.

9.31 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 *heapp, void *buf, size_t size)
  Initializes a memory heap from a static memory area.

• void * chHeapAllocAligned (memory_heap_t *heapp, size_t size, unsigned align)
  Allocates a block of memory from the heap by using the first-fit algorithm.

• void chHeapFree (void *p)
  Frees a previously allocated memory block.

• size_t chHeapStatus (memory_heap_t *heapp, size_t *totalp, size_t *largestp)
  Reports the heap status.

• static void * chHeapAlloc (memory_heap_t *heapp, size_t size)
  Allocates a block of memory from the heap by using the first-fit algorithm.

• static size_t chHeapGetSize (const void *p)
  Returns the size of an allocated block.

9.31.1 Detailed Description

Memory heaps macros and structures.

9.32 chmempools.c File Reference

Memory Pools code.

#include "ch.h"

Functions

• void chPoolObjectInitAligned (memory_pool_t *mp, size_t size, unsigned align, memgetfunc_t provider)
  Initializes an empty memory pool.

• void chPoolLoadArray (memory_pool_t *mp, void *p, size_t n)
  Loads a memory pool with an array of static objects.

• void * chPoolAllocI (memory_pool_t *mp)
  Allocates an object from a memory pool.

• void * chPoolAlloc (memory_pool_t *mp)
  Allocates an object from a memory pool.

• void chPoolFreeI (memory_pool_t *mp, void *objp)
  Releases an object into a memory pool.

• void chPoolFree (memory_pool_t *mp, void *objp)
  Releases an object into a memory pool.

• void chGuardedPoolObjectInitAligned (guarded_memory_pool_t *gmp, size_t size, unsigned align)
  Initializes an empty guarded memory pool.

• void chGuardedPoolLoadArray (guarded_memory_pool_t *gmp, void *p, size_t n)
  Loads a guarded memory pool with an array of static objects.

• void * chGuardedPoolAllocTimeoutS (guarded_memory_pool_t *gmp, sysinterval_t timeout)
  Allocates an object from a guarded memory pool.

• void * chGuardedPoolAllocTimeout (guarded_memory_pool_t *gmp, sysinterval_t timeout)
  Allocates an object from a guarded memory pool.

• void chGuardedPoolFree (guarded_memory_pool_t *gmp, void *objp)
  Releases an object into a guarded memory pool.
9.32.1 Detailed Description

Memory Pools code.

9.33 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 chPoolAddl (memory_pool_t *mp, void *objp)
  Adds an object to a memory pool.
- static void chGuardedPoolObjectInit (guarded_memory_pool_t *gmp, size_t size)
  Initializes an empty guarded memory pool.
- static cnt_t chGuardedPoolGetCounterI (guarded_memory_pool_t *gmp)
  Gets the count of objects in a guarded memory pool.
- static void *chGuardedPoolAlloc (guarded_memory_pool_t *gmp)
  Allocates an object from a guarded memory pool.
- static void chGuardedPoolFree (guarded_memory_pool_t *gmp, void *objp)
  Releases an object into a guarded memory pool.
- static void 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 chGuardedPoolAddl (guarded_memory_pool_t *gmp, void *objp)
  Adds an object to a guarded memory pool.
- static void chGuardedPoolAddS (guarded_memory_pool_t *gmp, void *objp)
  Adds an object to a guarded memory pool.

9.33.1 Detailed Description

Memory Pools macros and structures.

9.34 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 *chMsgPollS (void)
  Poll to check for an incoming message.
- void chMsgRelease (thread_t *tp, msg_t msg)
  Releases a sender thread specifying a response message.
9.34.1 Detailed Description

Messages code.

9.35 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.

9.35.1 Detailed Description

Messages macros and structures.

9.36 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.

9.36.1 Detailed Description

Mutexes code.

9.37 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 MUTEX_DECL(name) mutex_t name = __MUTEX_DATA(name)
  static mutex initializer.

Typedefs

- typedef struct ch_mutex mutex_t
  type of a mutex structure.
Functions

- void chMtxObjectInit (mutex_t *mp)
  Initializes a mutex structure.
- void chMtxLock (mutex_t *mp)
  Locks the specified mutex.
- void chMtxLockS (mutex_t *mp)
  Locks the specified mutex.
- bool chMtxTryLock (mutex_t *mp)
  Tries to lock a mutex.
- bool chMtxTryLockS (mutex_t *mp)
  Tries to lock a mutex.
- void chMtxUnlock (mutex_t *mp)
  Unlocks the specified mutex.
- void chMtxUnlockS (mutex_t *mp)
  Unlocks the specified mutex.
- void chMtxUnlockAll (void)
  Unlocks all mutexes owned by the invoking thread.
- void chMtxUnlockAllS (void)
  Unlocks all mutexes owned by the invoking thread.
- static bool chMtxQueueNotEmptyS (mutex_t *mp)
  Returns true if the mutex queue contains at least a waiting thread.
- static thread_t * chMtxGetOwnerI (mutex_t *mp)
  Returns the mutex owner thread.
- static mutex_t * chMtxGetNextMutexX (void)
  Returns the next mutex in the mutexes stack of the current thread.

9.37.1 Detailed Description

Mutexes macros and structures.

9.38 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 an objects_cache_t object.
- oc_object_t * chCacheGetObject (objects_cache_t *ocp, uint32_t group, uint32_t key)
Retrieves an object from the cache.

- void chCacheReleaseObject (objects_cache_t *ocp, oc_object_t *objp)

Releases an object into the cache.

- bool chCacheReadObject (objects_cache_t *ocp, oc_object_t *objp, bool async)

Reads object data from the storage.

- bool chCacheWriteObject (objects_cache_t *ocp, oc_object_t *objp, bool async)

Writes the object data back to storage.

### 9.38.1 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.

### 9.39 chobjcaches.h File Reference

Objects Caches macros and structures.
Data Structures

- struct ch_oc_hash_header
  Structure representing a hash table element.
- struct ch_oc_lru_header
  Structure representing an hash table element.
- struct ch_oc_object
  Structure representing a cached object.
- struct ch_objects_cache
  Structure representing a cache object.

Macros

Cached objects flags

- #define OC_FLAG_INLRU 0x00000001U
- #define OC_FLAG_INHASH 0x00000002U
- #define OC_FLAG_SHARED 0x00000004U
- #define OC_FLAG_NOTSYNC 0x00000008U
- #define OC_FLAG_LAZYWRITE 0x00000010U
- #define OC_FLAG_FORGET 0x00000020U

Typedefs

- typedef uint32_t oc_flags_t
  Flags of cached objects.
- typedef struct ch_oc_hash_header oc_hash_header_t
  Type of an hash element header.
- typedef struct ch_oc_lru_header oc_lru_header_t
  Type of an LRU element header.
- typedef struct ch_oc_object oc_object_t
  Type of a cached object.
- typedef struct ch_objects_cache objects_cache_t
  Type of a cache object.
- typedef bool(∗ oc_readf_t) (objects_cache_t ∗ocp, oc_object_t ∗objp, bool async)
  Object read function.
- typedef bool(∗ oc_writef_t) (objects_cache_t ∗ocp, oc_object_t ∗objp, bool async)
  Object write function.

Functions

- void chCacheObjectInit (objects_cache_t ∗ocp, ucnt_t hashn, oc_hash_header_t ∗hashp, ucnt_t objn, size_t objsz, void ∗objvp, oc_readf_t readf, oc_writef_t writef)
  Initializes a objects_cache_t object.
- oc_object_t ∗chCacheGetObject (objects_cache_t ∗ocp, uint32_t group, uint32_t key)
  Retrieves an object from the cache.
- void chCacheReleaseObjectI (objects_cache_t ∗ocp, oc_object_t ∗objp)
  Releases an object into the cache.
- bool chCacheReadObject (objects_cache_t ∗ocp, oc_object_t ∗objp, bool async)
  Reads object data from the storage.
- bool chCacheWriteObject (objects_cache_t ∗ocp, oc_object_t ∗objp, bool async)
  Writes the object data back to storage.
- static void chCacheReleaseObject (objects_cache_t ∗ocp, oc_object_t ∗objp)
  Releases an object into the cache.
9.39.1 Detailed Description

Objects Caches macros and structures.

9.40 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.

9.40.1 Detailed Description

Operating System Objects macros and structures.

9.41 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.
• 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.

### 9.41.1 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.

### 9.42 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.
9.42.1 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_USEPIPES` option must be enabled in `chconf.h`.

Note

Compatible with RT and NIL.

9.43 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.

9.43.1 Detailed Description

Pipes macros and structures.

9.44 chport.h File Reference

Port wrapper header.

#include "chcore.h"

9.44.1 Detailed Description

Port wrapper header.

9.45 chregistry.c File Reference

Threads registry code.

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

- `thread_t * chRegFirstThread (void)`
  Returns the first thread in the system.
- `thread_t * chRegNextThread (thread_t *tp)`
  Returns the thread next to the specified one.
- `thread_t * chRegFindThreadByName (const char *name)`
  Retrieves a thread pointer by name.
- `thread_t * chRegFindThreadByPointer (thread_t *tp)`
  Confirms that a pointer is a valid thread pointer.
- `thread_t * chRegFindThreadByWorkingArea (stkalign_t *wa)`
  Confirms that a working area is being used by some active thread.

9.45.1 Detailed Description

Threads registry code.

9.46 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.

9.46.1 Detailed Description

Threads registry macros and structures.

9.47 chrestrictions.h File Reference

Licensing restrictions header.

9.47.1 Detailed Description

Licensing restrictions header.

9.48 chrfcu.c File Reference

Runtime Faults Collection Unit code.

```c
#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.
9.48.1 Detailed Description

Runtime Faults Collection Unit code.

9.49 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.

9.49.1 Detailed Description

Runtime Faults Collection Unit macros and structures.
9.50  chsched.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 * chSchSelectFirstI (void)
  Makes runnable the fist thread in the ready list, does not reschedule internally.

9.50.1  Detailed Description

Scheduler code.

9.51  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_curthread() __instance_get_curthread(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 NOPRIOR (tprio_t)0`
  Ready list header priority.

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

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

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

- `#define HIGHPRIOR (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)`
  On a cond. variable.
- `#define CH_STATE_SLEEPING (tstate_t)`
  Sleeping.
- `#define CH_STATE_WTEXTIT (tstate_t)`
  Waiting a thread.
- `#define CH_STATE_WTOREVT (tstate_t)`
  One event.
- `#define CH_STATE_WTANDEVT (tstate_t)`
  Several events.
- `#define CH_STATE_SNDMSGQ (tstate_t)`
  Sending a message, in queue.
- `#define CH_STATE_SNDMSG (tstate_t)`
  Sent a message, waiting answer.
- `#define CH_STATE_WTMSG (tstate_t)`
  Waiting for a message.
- `#define CH_STATE_FINAL (tstate_t)`
  Thread terminated.
- `#define CH_STATE_NAMES`
  Thread states as array of strings.

Thread flags and attributes

- `#define CH_FLAG_MODE_MASK (tmode_t)U`
  Thread memory mode mask.
- `#define CH_FLAG_MODE_STATIC (tmode_t)U`
  Static thread.
- `#define CH_FLAG_MODE_HEAP (tmode_t)U`
  Thread allocated from a Memory Heap.
- `#define CH_FLAG_MODE_MPOOL (tmode_t)U`
  Thread allocated from a Memory Pool.
- `#define CH_FLAG_TERMINATE (tmode_t)U`
  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)`
  ChibiOS/RT
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 * chSchSelectFirstI (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.

9.51.1 Detailed Description
Scheduler macros and structures.

9.52 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.
• msg_t chSemSignalWait (semaphore_t *sps, semaphore_t *spw)
  Performs atomic signal and wait operations on two semaphores.
9.52.1 Detailed Description

Semaphores code.

9.53 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)
Adds the specified value to the semaphore counter.
• msg_t chSemSignalWait (semaphore_t *sps, semaphore_t *spw)
  Performs atomic signal and wait operations on two semaphores.
• static void chSemReset (semaphore_t *sp, cnt_t n)
  Performs a reset operation on the semaphore.
• static void chSemResetI (semaphore_t *sp, cnt_t n)
  Performs a reset operation on the semaphore.
• static void chSemFastWaitI (semaphore_t *sp)
  Decreases the semaphore counter.
• static void chSemFastSignalI (semaphore_t *sp)
  Increases the semaphore counter.
• static cnt_t chSemGetCounterI (const semaphore_t *sp)
  Returns the semaphore counter current value.

9.53.1 Detailed Description

Semaphores macros and structures.

9.54 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.

9.54.1 Detailed Description

Statistics module code.
9.55  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.

9.55.1  Detailed Description

Statistics module macros and structures.

9.56  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.
- bool chSysIntegrityCheckl (unsigned testmask)
  System integrity check.
- void chSysTimerHandlerl (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.

9.56.1 Detailed Description

System related code.

9.57 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 entry 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_1)(((freq) + 999UL) / 1000UL) * (msec)` 
  Milliseconds to realtime counter.
- `#define US2RTC(freq, usec) (rtcnt_1)(((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.
- 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.
- static thread_t * chSysIdleThreadX (void)
  Returns a pointer to the idle thread.

9.57.1 Detailed Description

System related macros and structures.

9.58 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_memfill (uint8_t * startp, uint8_t * endp, uint8_t v)`
  - Memory 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 * 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.

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

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

\section*{9.58.1 Detailed Description}

Threads code.

\section*{9.59 \texttt{chthreads.h} File Reference}

Threads module macros and structures.

**Data Structures**

- \texttt{struct thread_descriptor_t}
  
  Type of a thread descriptor.

**Macros**

**Threads queues**

- \texttt{#define \_THREADS\_QUEUE\_DATA(name) \{\_CH\_QUEUE\_DATA(name)\}}
  
  Data part of a static threads queue object initializer.

- \texttt{#define THREADS\_QUEUE\_DECL(name) threads_queue_t name = \_THREADS\_QUEUE\_DATA(name)}
  
  Static threads queue object initializer.

**Working Areas**

- \texttt{#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.

- \texttt{#define THD\_WORKING\_AREA(s, n) PORT\_WORKING\_AREA(s, n)}
  
  Static working area allocation.

- \texttt{#define THD\_WORKING\_AREA\_BASE(s) ((stkalign_t *)(s))}
  
  Base of a working area casted to the correct type.

- \texttt{#define THD\_WORKING\_AREA\_END(s)}
  
  End of a working area casted to the correct type.

**Threads abstraction macros**

- \texttt{#define THD\_FUNCTION(tname, arg) PORT\_THD\_FUNCTION(tname, arg)}
  
  Thread declaration macro.

**Threads initializers**

- \texttt{#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.

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_memfill (uint8_t *startp, uint8_t *endp, uint8_t v)
  Memory fill utility.
• thread_t *chThdCreateSuspendedI (const thread_descriptor_t *tdp)
  Creates a new thread into a static memory area.
• thread_t *chThdCreateSuspended (const thread_descriptor_t *tdp)
  Creates a new thread into a static memory area.
• thread_t *chThdCreateI (const thread_descriptor_t *tdp)
  Creates a new thread into a static memory area.
• thread_t *chThdCreate (const thread_descriptor_t *tdp)
  Creates a new thread into a static memory area.
• thread_t *chThdCreateStatic (void *wsp, size_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)
9.59 chthreads.h File Reference

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 chThdQueuesIsEmptyI (threads_queue_t *tqp)
  Evaluates to true if the specified queue is empty.

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

9.59.1 Detailed Description

Threads module macros and structures.
9.60  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_INFINITY ((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.
9.61  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 chTMChainMeasurementToObject (time_measurement_t *tmp1, time_measurement_t *tmp2)
  Stops a measurement and chains to the next one using the same time stamp.
9.61.1 Detailed Description

Time Measurement module code.

9.62 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.

9.62.1 Detailed Description

Time Measurement module macros and structures.

9.63 chtrace.c File Reference

Tracer code.

#include "ch.h"
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 a 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.

9.63.1 Detailed Description

Tracer code.

9.64 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
  Trace buffer entries.
- #define CH_DBG_TRACE_BUFFER_SIZE 128
  Trace buffer entries.

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.
9.65.1 Detailed Description

Tracer macros and structures.

9.65 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 "21.6.0"`
  ChibiOS version string.
- `#define CH_VERSION_YEAR 21`
  ChibiOS version release year.
- `#define CH_VERSION_MONTH 6`
  ChibiOS version release month.
- `#define CH_VERSION_PATCH 0`
  ChibiOS version patch number.
- `#define CH_VERSION_NICKNAME "Atrani"`
  ChibiOS version nickname.

9.65.1 Detailed Description

Version Module macros and structures.

9.66 chvt.c File Reference

Time and Virtual Timers module code.

#include "ch.h"
Functions

- 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, systime_t now, 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.

9.66.1 Detailed Description

Time and Virtual Timers module code.

9.67 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.

- static void chVTObjectInit (virtual_timer_t ∗vtp)  
  Initializes a virtual_timer_t object.
• static systime_t chVTGetSystemTimeX (void)
  Current system time.
• static systime_t chVTGetSystemTime (void)
  Current system time.
• static sysinterval_t chVTTimeElapsedSinceX (systime_t start)
  Returns the elapsed time since the specified start time.
• static bool chVTIsSystemTimeWithinX (systime_t start, systime_t end)
  Checks if the current system time is within the specified time window.
• static bool chVTIsSystemTimeWithin (systime_t start, systime_t end)
  Checks if the current system time is within the specified time window.
• static bool chVTGetTimersStateI (sysinterval_t ∗timep)
  Returns the time interval until the next timer event.
• static bool chVTIsArmedI (const virtual_timer_t ∗vtp)
  Returns true if the specified timer is armed.
• static bool chVTIsArmed (const virtual_timer_t ∗vtp)
  Returns true if the specified timer is armed.
• static void chVTResetI (virtual_timer_t ∗vtp)
  Disables a Virtual Timer.
• static void chVTReset (virtual_timer_t ∗vtp)
  Disables a Virtual Timer.
• static void chVTSetI (virtual_timer_t ∗vtp, sysinterval_t delay, vtfunc_t vtfunc, void ∗par)
  Enables a 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.

9.67.1 Detailed Description

Time and Virtual Timers module macros and structures.
Index

__BSEMAPHORE_DATA__
  Binary Semaphores, 354

__CHIBIOS_OSLIB__
  Version Numbers and Identification, 350

__CHIBIOS_RT__
  Version Numbers and Identification, 29

__CHIBIOS__
  Release Information, 22

__CH_OFFSETOF__
  OS Types and Structures, 60

__CH_QUEUE_DATA__
  Lists and Queues, 75

__CH_STRINGIFY__
  OS Types and Structures, 59

__CH_USED__
  OS Types and Structures, 60

__CONDVAR_DATA__
  Condition Variables, 262

__EVENTSOURCE_DATA__
  Event Flags, 275

__GUARDEDMEMORYPOOL_DATA__
  Memory Pools, 439

__MAILBOX_DATA__
  Mailboxes, 366

__MEMORYPOOL_DATA__
  Memory Pools, 438

__MUTEX_DATA__
  Mutexes, 247

__PIPE_DATA__
  Pipes, 383

__SEMAPHORE_DATA__
  Counting Semaphores, 229

__THREADS_QUEUE_DATA__
  Threads, 183

__ch_delegate_fn0__
  Delegate Threads, 392

__ch_delegate_fn1__
  Delegate Threads, 393

__ch_delegate_fn2__
  Delegate Threads, 393

__ch_delegate_fn3__
  Delegate Threads, 393

__ch_delegate_fn4__
  Delegate Threads, 395

__core_init__
  Core Memory Manager, 422

__dbg_check_disable__
  Checks and Assertions, 328

__dbg_check_enable__
  Checks and Assertions, 329

__dbg_check_enter_isr__
  Checks and Assertions, 332

__dbg_check_leave_isr__
  Checks and Assertions, 332

__dbg_check_lock__
  Checks and Assertions, 330

__dbg_check_lock_from_isr__
  Checks and Assertions, 331

__dbg_check_suspend__
  Checks and Assertions, 329

__dbg_check_unlock__
  Checks and Assertions, 330

__dbg_check_unlock_from_isr__
  Checks and Assertions, 331

__dbg_object_init__
  Checks and Assertions, 334

__factory_init__
  Dynamic Objects Factory, 496

__heap_init__
  Memory Heaps, 432

__idle_thread__
  OS Instances, 69

__instance_get_currthread__
  OS Instances, 68

__instance_set_currthread__
  OS Instances, 68

__oslib_init__
  Version Numbers and Identification, 351

__reg_object_init__
  Registry, 321

__rfcu_object_init__
  Runtime Faults Collection Unit, 73

__sch_get_currthread__
  Scheduler, 97

__sch_ready-ahead__
  Scheduler, 99

__sch_ready-behind__
  Scheduler, 97

__sch_reschedule-ahead__
  Scheduler, 100

__sch_reschedule-behind__
  Scheduler, 100

__stats_ctxswc__
  Statistics, 345

__stats_increase_irq__
  Statistics, 345

__stats_object_init__
  Statistics, 347
buf_list ch_objects_factory, 537
buffer mailbox_t, 591
pipe_t, 599
trace_buffer_t, 609
cache_sem ch_objects_cache, 534
ch.h, 615
ch0 System Management, 134
ch1 System Management, 135
ch_binary_semaphore, 515
CH_CFG_CONTEXT_SWITCH_HOOK Options, 51
CH_CFG_FACTORY_GENERIC_BUFFERS Dynamic Objects Factory, 493 Options, 45
CH_CFG_FACTORY_MAILBOXES Dynamic Objects Factory, 494 Options, 46
CH_CFG_FACTORY_MAX_NAMES_LENGTH Dynamic Objects Factory, 493 Options, 45
CH_CFG_FACTORY_OBJECTS_REGISTRY Dynamic Objects Factory, 494 Options, 46
CH_CFG_FACTORY_OBJECTS_REGISTRY Dynamic Objects Factory, 493 Options, 45
CH_CFG_FACTORY_PIPES Dynamic Objects Factory, 494, 495 Options, 46
CH_CFG_FACTORY_SEMAPHORES Dynamic Objects Factory, 493 Options, 45
CH_CFG_IDLE_ENTER_HOOK Options, 51
CH_CFG_IDLE_LEAVE_HOOK Options, 52
CH_CFG_IDLE_LOOP_HOOK Options, 52
CH_CFG_INTERVALS_SIZE Options, 36
CH_CFG_IRQ_EPILOGUE_HOOK Options, 51
CH_CFG_IRQ_PROLOGUE_HOOK Options, 51
CH_CFG_MEMCORE_SIZE Core Memory Manager, 420 Options, 42
CH_CFG_NO_IDLE_THREAD Options, 37
CH_CFG_OPTIMIZE_SPEED Options, 37
CH_CFG_OS_INSTANCE_EXTRA_FIELDS Options, 49
CH_CFG_OS_INSTANCE_INIT_HOOK
Options, 49
CH_CFG_RUNTIME_FAULTS_HOOK
Options, 53
CH_CFG_SMP_MODE
Options, 35
CH_CFG_ST_FREQUENCY
Options, 36
CH_CFG_ST_RESOLUTION
Options, 35
CH_CFG_ST_TIMEDELTA
Options, 36
CH_CFG_SYSTEM_EXTRA_FIELDS
Options, 49
CH_CFG_SYSTEM_HALT_HOOK
Options, 53
CH_CFG_SYSTEM_INIT_HOOK
Options, 49
CH_CFG_SYSTEM_TICK_HOOK
Options, 52
CH_CFG_THREAD_EXIT_HOOK
Options, 50
CH_CFG_THREAD_EXTRA_FIELDS
Options, 50
CH_CFG_THREAD_INIT_HOOK
Options, 50
CH_CFG_TIME_QUANTUM
Options, 37
CH_CFG_TIME_TYPES_SIZE
Options, 36
CH_CFG_TRACE_HOOK
Options, 53
CH_CFG_USE_CONDVARS
Options, 39
CH_CFG_USE_CONDVARS_TIMEOUT
Options, 40
CH_CFG_USE_DELEGATES
Options, 44
CH_CFG_USE_DYNAMIC
Options, 41
CH_CFG_USE_EVENTS
Options, 40
CH_CFG_USE_EVENTS_TIMEOUT
Options, 40
CH_CFG_USE_FACTORY
Options, 45
CH_CFG_USE_HEAP
Options, 43
CH_CFG_USE_JOBS
Options, 44
CH_CFG_USE_MAILBOXES
Options, 42
CH_CFG_USE_MEMCORE
Options, 42
CH_CFG_USE_MEMPOOLS
Options, 43
CH_CFG_USE_MESSAGES
Options, 41
CH_CFG_USE_MESSAGES_PRIORITY
Options, 41
CH_CFG_USE_MUTEXES
Options, 39
CH_CFG_USE_MUTEXES_RECURSIVE
Options, 39
CH_CFG_USE_OBJ_CACHES
Options, 39
CH_CFG_USE_OBJ_FIFOS
Options, 43
CH_CFG_USE_PIPES
Options, 44
CH_CFG_USE_REGISTERS
Options, 38
CH_CFG_USE_SEMAPHORES
Options, 38
CH_CFG_USE_SEMAPHORES_PRIORITY
Options, 39
CH_CFG_USE_TIMESTAMP
Options, 38
ch_core0_cfg
System Management, 135
ch_core1_cfg
System Management, 135
CH_CUSTOMER_ID_CODE
Customer Information, 25
CH_CUSTOMER_ID_STRING
Customer Information, 24
CH_CUSTOMER_LICENSE_EOS_DATE
Customer Information, 25
CH_CUSTOMER_LICENSE_VERSION_DATE
Customer Information, 25
CH_CUSTOMER_LICENSE_VERSION_MONTH
Customer Information, 25
CH_CUSTOMER_LICENSE_VERSION_YEAR
Customer Information, 25
CH_DBG_ENABLE_ASSERTS
Options, 47
CH_DBG_ENABLE_CHECKS
Options, 47
CH_DBG_ENABLE_STACK_CHECK
Options, 48
CH_DBG_FILL_THREADS
Options, 48
CH_DBG_STACK_FILL_VALUE
Checks and Assertions, 327
CH_DBG_STATISTICS
Options, 46
CH_DBG_SYSTEM_STATE_CHECK
Options, 46
CH_DBG_THREADS_PROFILING
Options, 48
CH_DBG_TRACE_BUFFER_SIZE
Options, 47
Tracing, 336

CH_DBG_TRACE_MASK
Options, 47
Tracing, 336

ch_delta_list, 516
delta, 517
next, 517
prev, 517

ch_delta_list_t
Lists and Queues, 76

ch_dlist_dequeue
Lists and Queues, 88

ch_dlist_init
Lists and Queues, 84

ch_dlist_insert
Lists and Queues, 87

ch_dlist_insert_after
Lists and Queues, 86

ch_dlist_insert_before
Lists and Queues, 86

ch_dlist_isempty
Lists and Queues, 84

ch_dlist_isfirst
Lists and Queues, 85

ch_dlist_islast
Lists and Queues, 85

ch_dlist_notempty
Lists and Queues, 85

ch_job_descriptor, 526
jobarg, 527
jobfunc, 527

ch_jobs_queue, 527
free, 528
mbx, 529

CH_KERNEL_MAJOR
Version Numbers and Identification, 30

CH_KERNEL_MINOR
Version Numbers and Identification, 30

CH_KERNEL_PATCH
Version Numbers and Identification, 30

CH_KERNEL_STABLE
Version Numbers and Identification, 29

CH_KERNEL_VERSION
Version Numbers and Identification, 30

CH_LICENSE
License Settings, 26

CH_LICENSE_FEATURES
License Settings, 26

CH_LICENSE_MAX_DEPLOY
License Settings, 26

CH_LICENSE_MAGIC
License Settings, 26

CH_LICENSE_MODIFIABLE_CODE
License Settings, 26

CH_LICENSE_TYPE_STRING
License Settings, 26

ch_list, 529
next, 529

ch_list_init
Lists and Queues, 77

ch_list_isempty
Lists and Queues, 77

ch_list_link
Lists and Queues, 77

ch_list_notempty
Lists and Queues, 77
ch_list_t
  Lists and Queues, 76
ch_list_unlink
  Lists and Queues, 78
ch_memcore
  Core Memory Manager, 429
ch_mutex, 530
cnt, 531
next, 531
owner, 531
queue, 531
ch_objects_cache, 532
cache_sem, 534
hashn, 533
hashp, 533
lru, 534
lru_sem, 534
objn, 533
objsz, 534
objvp, 534
readf, 534
writef, 535
ch_objects_factory, 535
buf_list, 537
fifo_list, 537
mbx_list, 537
mtx, 536
obj_list, 536
obj_pool, 536
pipe_list, 537
sem_list, 537
sem_pool, 537
ch_objects_fifo, 538
free, 539
mbx, 539
ch_oc_hash_header, 539
hash_next, 540
hash_prev, 541
ch_oc_lru_header, 541
hash_next, 542
hash_prev, 542
lru_next, 542
lru_prev, 542
ch_oc_object, 543
dptr, 545
hash_next, 544
hash_prev, 544
lru_next, 544
lru_prev, 544
obj_flags, 545
obj_group, 544
obj_key, 545
obj_sem, 545
ch_os_instance, 546
config, 548
core_id, 547
dbg, 548
kernel_stats, 548
mainthread, 548
reglist, 547
rfcu, 547
rlist, 547
trace_buffer, 548
tvlist, 547
ch_os_instance_config, 549
idlethread_base, 550
idlethread_end, 550
mainthread_base, 549
mainthread_end, 550
name, 549
CH_OSLIB_MAJOR
  Version Numbers and Identification, 351
CH_OSLIB_MINOR
  Version Numbers and Identification, 351
CH_OSLIB_PATCH
  Version Numbers and Identification, 351
CH_OSLIB_STABLE
  Version Numbers and Identification, 350
CH_OSLIB_VERSION
  Version Numbers and Identification, 350
ch_pqueue_init
  Lists and Queues, 82
ch_pqueue_insert_ahead
  Lists and Queues, 83
ch_pqueue_insert_behind
  Lists and Queues, 83
ch_pqueue_remove_highest
  Lists and Queues, 82
ch_priority_queue, 550
next, 551
prev, 551
prio, 551
ch_priority_queue_t
  Lists and Queues, 76
ch_queue, 552
next, 552
prev, 552
CH_QUEUE_DECL
  Lists and Queues, 76
ch_queue_dequeue
  Lists and Queues, 81
ch_queue_fifo_remove
  Lists and Queues, 80
ch_queue_init
  Lists and Queues, 80
ch_queue_insert
  Lists and Queues, 80
ch_queueIsEmpty
  Lists and Queues, 79
ch_queue_fifo_remove
  Lists and Queues, 81
ch_queue_notempty
  Lists and Queues, 80
ch_queue_t
  Lists and Queues, 76
ch_ready_list, 553

ChibiOS/RT
current, 554
queue, 554
ch_registered_static_object, 554
  element, 555
  objp, 555
ch_registry, 556
queue, 556
ch_rfcu, 557
  mask, 557
CH_RFCU_ALL_FAULTS
  Runtime Faults Collection Unit, 71
ch_sch_prio_insert
  Scheduler, 101
ch_semaphore, 558
cnt, 559
queue, 559
CH_STATE_CURRENT
  Scheduler, 93
CH_STATE_FINAL
  Scheduler, 95
CH_STATE_NAMES
  Scheduler, 96
CH_STATE_QUEUED
  Scheduler, 94
CH_STATE_READY
  Scheduler, 93
CH_STATE_SLEEPING
  Scheduler, 94
CH_STATE SNDMSG
  Scheduler, 95
CH_STATE SNDMSGQ
  Scheduler, 95
CH_STATE_SUSPENDED
  Scheduler, 93
CH_STATE_WTANDEVT
  Scheduler, 95
CH_STATE_WTCOND
  Scheduler, 94
CH_STATE_WTEXIT
  Scheduler, 94
CH_STATE_WTMSG
  Scheduler, 95
CH_STATE_WTMTX
  Scheduler, 94
CH_STATE_WTOREVT
  Scheduler, 95
CH_STATE_WTSTART
  Scheduler, 93
CH_SYS_CORE0_MEMORY
  System Management, 113
CH_SYS_CORE1_MEMORY
  System Management, 113
ch_system, 559
  instances, 561
  reglist, 561
  rfcu, 561
state, 560
System Management, 134
tmc, 561
ch_system_debug, 562
  isr_cnt, 562
  lock_cnt, 563
  panic_msg, 562
ch_system_t
  OS Types and Structures, 65
ch_thread, 563
ctx, 566
  pending, 570
  ewmask, 569
  exitcode, 568
  flags, 567
  hdr, 565
  list, 565
  mpool, 571
  msgqueue, 570
  mtxlist, 570
  name, 566
  owner, 566
  pqueue, 565
  queue, 565
  rdymsg, 567
  realprio, 570
  refs, 567
  rqueue, 566
  sentmsg, 568
  state, 566
  stats, 571
  ticks, 567
  time, 567
  u, 569
  wabase, 566
  waiting, 570
  wtmtxp, 569
  wtobjp, 568
  wtsemp, 569
  wttrp, 568
ch_threads_queue, 571
  queue, 572
CH_VERSION
  Release Information, 22
CH_VERSION_DATE
  Release Information, 23
CH_VERSION_MONTH
  Release Information, 23
CH_VERSION_NICKNAME
  Release Information, 23
CH_VERSION_PATCH
  Release Information, 23
CH_VERSION_STABLE
  Release Information, 22
CH_VERSION_YEAR
  Release Information, 23
ch_virtual_timer, 572
dlist, 573
chbiostm.rs
chDelegateCallDirect0
Delegate Threads, 397
chDelegateCallDirect1
Delegate Threads, 398
chDelegateCallDirect2
Delegate Threads, 399
chDelegateCallDirect3
Delegate Threads, 400
chDelegateCallDirect4
Delegate Threads, 401
chDelegateCallVeneer
Delegate Threads, 395
chDelegateDispatch
Delegate Threads, 396
chDelegateDispatchTimeout
Delegate Threads, 396
chdelegates.c, 626
chdelegates.h, 627
chdynamic.c, 628
chdynamic.h, 628
chearly.h, 629
Checks, 54
Checks and Assertions, 325
__dbg_check_disable, 328
__dbg_check_enable, 329
__dbg_check_enter_isr, 332
__dbg_check_leave_isr, 332
__dbg_check_lock, 330
__dbg_check_lock_from_isr, 331
__dbg_check_suspend, 329
__dbg_check_unlock, 330
__dbg_check_unlock_from_isr, 331
__dbg_object_init, 334
CH_DBG_STACK_FILL_VALUE, 327
chDbgAssert, 327
chDbgCheck, 327
chDbgCheckClassI, 333
chDbgCheckClassS, 333
system_debug_t, 328
chevents.c, 630
chevents.h, 631
chEvtAddEvents
Event Flags, 281
chEvtAddEventsI
Event Flags, 298
chEvtBroadcast
Event Flags, 297
chEvtBroadcastFlags
Event Flags, 287
chEvtBroadcastFlagsI
Event Flags, 286
chEvtBroadcastI
Event Flags, 297
chEvtDispatch
Event Flags, 288
chEvtGetAndClearEvents
Event Flags, 280
chEvtGetAndClearEventsI
Event Flags, 279
chEvtGetAndClearFlags
Event Flags, 283
chEvtGetAndClearFlagsI
Event Flags, 282
chEvtGetEventsX
Event Flags, 298
chEvtIsListeningI
Event Flags, 296
chEvtObjectInit
Event Flags, 294
chEvtRegister
Event Flags, 295
chEvtRegisterMask
Event Flags, 295
chEvtRegisterMaskWithFlags
Event Flags, 277
chEvtRegisterMaskWithFlagsI
Event Flags, 276
chEvtSignal
Event Flags, 285
chEvtSignall
Event Flags, 284
chEvtUnregister
Event Flags, 278
chEvtWaitAll
Event Flags, 290
chEvtWaitAllTimeout
Event Flags, 293
chEvtWaitAny
Event Flags, 289
chEvtWaitAnyTimeout
Event Flags, 292
chEvtWaitOne
Event Flags, 288
chEvtWaitOneTimeout
Event Flags, 291
chfactory.c, 633
chfactory.h, 634
chFactoryCreateBuffer
Dynamic Objects Factory, 499
chFactoryCreateMailbox
Dynamic Objects Factory, 503
chFactoryCreateObjectsFIFO
Dynamic Objects Factory, 504
chFactoryCreatePipe
Dynamic Objects Factory, 506
chFactoryCreateSemaphore
Dynamic Objects Factory, 501
chFactoryDuplicateReference
Dynamic Objects Factory, 509
chFactoryFindBuffer
Dynamic Objects Factory, 500
chFactoryFindMailbox
Dynamic Objects Factory, 503
chFactoryFindObject
Dynamic Objects Factory, 498
chFactoryFindObjectByPointer
Dynamic Objects Factory, 498
INDEX

Jobs Queues, 405
chJobGet
Jobs Queues, 406
chJobGetTimeout
Jobs Queues, 408
chJobGetTimeoutS
Jobs Queues, 407
chJobObjectInit
Jobs Queues, 404
chJobPost
Jobs Queues, 411
chJobPostAhead
Jobs Queues, 414
chJobPostAheadI
Jobs Queues, 412
chJobPostAheadS
Jobs Queues, 413
chJobPostI
Jobs Queues, 409
chJobPostS
Jobs Queues, 410
chjobs.h, 637
chlib.h, 639
chlicense.h, 640
chlists.h, 641
chMBFetchI
Mailboxes, 378
chMBFetchTimeout
Mailboxes, 376
chMBFetchTimeoutS
Mailboxes, 377
chMBGetFreeCountI
Mailboxes, 380
chMBGetSizeI
Mailboxes, 379
chMBGetUsedCountI
Mailboxes, 380
chMBObjectInit
Mailboxes, 367
chmbboxes.c, 643
chmbboxes.h, 643
chMBPeekI
Mailboxes, 381
chMBPostAheadI
Mailboxes, 375
chMBPostAheadTimeout
Mailboxes, 373
chMBPostAheadTimeoutS
Mailboxes, 374
chMBPostI
Mailboxes, 372
chMBPostTimeout
Mailboxes, 370
chMBPostTimeoutS
Mailboxes, 371
chMBReset
Mailboxes, 368
chMBResetI
Mailboxes, 369
chMBResumeX
Mailboxes, 382
chmemcore.c, 645
chmemcore.h, 645
chmemheaps.c, 646
chmemheaps.h, 647
chmempools.c, 648
chmempools.h, 649
chmsg.c, 650
chmsg.h, 651
chMsgGet
Synchronous Messages, 309
chMsgIsPendingI
Synchronous Messages, 308
chMsgPoll
Synchronous Messages, 307
chMsgPollS
Synchronous Messages, 303
chMsgRelease
Synchronous Messages, 304
chMsgReleaseS
Synchronous Messages, 301
chMsgSend
Synchronous Messages, 310
chMsgWait
Synchronous Messages, 305
chMsgWaitS
Synchronous Messages, 301
chMsgWaitTimeout
Synchronous Messages, 306
chMsgWaitTimeoutS
Synchronous Messages, 302
chmtx.c, 651
chmtx.h, 652
chMtxGetNextMutexX
Mutexes, 260
chMtxGetOwnerI
Mutexes, 259
chMtxLock
Mutexes, 250
chMtxLockS
Mutexes, 251
chMtxObjectInit
Mutexes, 249
chMtxQueueNotEmptyS
Mutexes, 258
chMtxTryLock
Mutexes, 252
chMtxTryLockS
Mutexes, 253
chMtxUnlock
Mutexes, 254
chMtxUnlockAll
Mutexes, 257
chMtxUnlockAllS
Mutexes, 256
chMtxUnlockS
INDEX

Mutexes, 255
chobjcaches.c, 653
chobjcaches.h, 654
chobjects.h, 656
chobjfiltos.h, 657
chPipeGetFreeCount
Pipes, 389
chPipeGetSize
Pipes, 388
chPipeGetUsedCount
Pipes, 388
chPipeObjectInit
Pipes, 386
chPipeReadTimeout
Pipes, 387
chPipeReset
Pipes, 386
chPipeResume
Pipes, 389
chpipes.c, 658
chpipes.h, 659
chPipeWriteTimeout
Pipes, 386
chPoolAdd
Memory Pools, 452
chPoolAddI
Memory Pools, 453
chPoolAlloc
Memory Pools, 443
chPoolAllocI
Memory Pools, 442
chPoolFree
Memory Pools, 445
chPoolFreeI
Memory Pools, 444
chPoolLoadArray
Memory Pools, 441
chPoolObjectInit
Memory Pools, 451
chPoolObjectInitAligned
Memory Pools, 440
chport.h, 660
chRegFindThreadByName
Registry, 318
chRegFindThreadByPointer
Registry, 319
chRegFindThreadByWorkingArea
Registry, 320
chRegFirstThread
Registry, 316
chRegGetThreadByNameX
Registry, 322
chregistry.c, 660
chregistry.h, 661
chRegNextThread
Registry, 317
chRegSetThreadName
Registry, 321
chRegSetThreadNameX
Registry, 322
chrestrictions.h, 662
chrfcu.c, 662
chrfcu.h, 663
chRFCUCollectFaultsI
Runtime Faults Collection Unit, 72
chRFCUGetAndClearFaultsI
Runtime Faults Collection Unit, 72
chschd.c, 664
chschd.h, 664
chSchDoPreemption
Scheduler, 107
chSchDoYieldS
Scheduler, 108
chSchGoSleepS
Scheduler, 108
chSchGoSleepTimeoutS
Scheduler, 103
chSchIsPreemptionRequired
Scheduler, 104
chSchPreemption
Scheduler, 108
chSchReadyI
Scheduler, 102
chSchRescheduleS
Scheduler, 106
chSchSelectFirstI
Scheduler, 109
chSchWakeupS
Scheduler, 105
chsem.c, 667
chsem.h, 668
chSemAddCounterI
Counting Semaphores, 239
chSemFastSignall
Counting Semaphores, 243
chSemFastWaitI
Counting Semaphores, 240
chSemGetCounter
Counting Semaphores, 244
chSemObjectInit
Counting Semaphores, 230
chSemReset
Counting Semaphores, 241
chSemResetI
Counting Semaphores, 242
chSemResetWithMessage
Counting Semaphores, 243
chSemResetWithMessageI
Counting Semaphores, 231
chSemSignal
Counting Semaphores, 232
chSemSignal
Counting Semaphores, 237
chSemSignalWait
Counting Semaphores, 238
chSemSleep
Counting Semaphores, 240
chSemWait
chThdSleepSeconds
  Threads, 187
chThdSleepUntil
  Threads, 205
chThdSleepUntilWindowed
  Threads, 206
chThdStart
  Threads, 196
chThdStartI
  Threads, 218
chThdSuspendS
  Threads, 208
chThdSuspendTimeoutS
  Threads, 209
chThdTerminate
  Threads, 203
chThdTerminatedX
  Threads, 217
chThdWait
  Threads, 201
chThdYield
  Threads, 207
cthreads.c, 673
cthreadsh.h, 675
ctime.h, 678
chTimeAddX
  Time and Intervals, 149
chTimeDiffX
  Time and Intervals, 150
chTime2MS
  Time and Intervals, 148
chTime2S
  Time and Intervals, 148
chTime2US
  Time and Intervals, 149
chTimeIsInRangeX
  Time and Intervals, 150
chTimeMS2I
  Time and Intervals, 147
chTimeS2I
  Time and Intervals, 146
chTimeStampAddX
  Time and Intervals, 151
chTimeStampDiffX
  Time and Intervals, 151
chTimeStampIsInRangeX
  Time and Intervals, 152
chTimeUS2I
  Time and Intervals, 147
chtm.c, 679
chtm.h, 680
chTMChainMeasurementToX
  Time Measurement, 225
chTMObjectInit
  Time Measurement, 223
chTMStartMeasurementX
  Time Measurement, 224
chTMStopMeasurementX
  Time Measurement, 224
chtrace.c, 680
chtrace.h, 681
chTraceResume
  Tracing, 343
chTraceResumeI
  Tracing, 343
chTraceSuspend
  Tracing, 342
chTraceSuspendI
  Tracing, 341
chTraceWrite
  Tracing, 340
chTraceWriteI
  Tracing, 339
chvts.h, 683
chvt.c, 683
chvt.h, 684
chVTDoReset
  Virtual Timers, 157
chVTDoSetContinuousI
  Virtual Timers, 156
chVTDoSet
  Virtual Timers, 155
chVTDoTickI
  Virtual Timers, 159
chVTGetReloadIntervalX
  Virtual Timers, 175
chVTGetRemainingIntervalI
  Virtual Timers, 158
chVTGetSystemTime
  Virtual Timers, 162
chVTGetSystemTimeX
  Virtual Timers, 162
chVTGetTimersStateI
  Virtual Timers, 166
chVTGetTimeStamp
  Virtual Timers, 176
chVTGetTimeStampI
  Virtual Timers, 160
chVTIsArmed
  Virtual Timers, 168
chVTIsArmedI
  Virtual Timers, 167
chVTIsSystemTimeWithin
  Virtual Timers, 165
chVTIsSystemTimeWithinX
  Virtual Timers, 164
chVTOBJECTINIT
  Virtual Timers, 161
chVTReset
  Virtual Timers, 170
chVTResetI
  Virtual Timers, 169
chVTResetTimeStamp
  Virtual Timers, 177
chVTResetTimeStampI
  Virtual Timers, 161
chVTSet
Virtual Timers, 172
chVTSetContinuous
Virtual Timers, 174
chVTSetContinuousI
Virtual Timers, 173
chVTSet
Virtual Timers, 171
chVTSetReloadIntervalX
Virtual Timers, 175
chVTTimeElapsedSinceX
Virtual Timers, 163
cmtx
pipe_t, 600
cnt
ch_mutex, 531
ch_semaphore, 559
mailbox_t, 592
pipe_t, 600
cnt_t
OS Types and Structures, 63
Complex Services, 462
Condition Variables, 261
__CONDVAR_DATA, 262
chCondBroadcast, 265
chCondBroadcastI, 266
chCondObjectInit, 262
chCondSignal, 263
chCondSignalI, 264
chCondWait, 267
chCondWaitS, 268
chCondWaitTimeout, 269
chCondWaitTimeoutS, 270
condition_variable_t, 262
CONDVAR_DECL, 262
condition_variable, 580
queue, 581
condition_variable_t
Condition Variables, 262
CONDVAR_DECL
Condition Variables, 262
config
ch_os_instance, 548
Configuration, 31
Core Memory Manager, 419
__core_init, 422
CH_CFG_MEMCORE_SIZE, 420
ch_memcore, 429
chCoreAlloc, 428
chCoreAllocAligned, 427
chCoreAllocAlignedI, 426
chCoreAllocAlignedWithOffset, 421
chCoreAllocAlignedWithOffsetI, 420
chCoreAllocFromBase, 424
chCoreAllocFromBaseI, 422
chCoreAllocFromTop, 425
chCoreAllocFromTopI, 423
chCoreAllocI, 428
chCoreGetStatusX, 426
memgetfunc2_t, 421
memgetfunc_t, 421
core_id
ch_os_instance, 547
core_id_t
OS Types and Structures, 63
Counting Semaphores, 228
__SEMAPHORE_DATA, 229
chSemAddCounterI, 239
chSemFastSignall, 243
chSemFastWaitI, 243
chSemGetCounterI, 244
chSemObjectInit, 230
chSemReset, 241
chSemResetI, 242
chSemResetWithMessage, 231
chSemResetWithMessageI, 232
chSemSignal, 237
chSemSignall, 238
chSemSignalWait, 240
chSemWait, 233
chSemWaitS, 234
chSemWaitI, 235
chSemWaitTimeout, 236
chSemWaitTimeoutS, 236
SEMAPHORE_DECL, 230
semaphore_t, 230
ctx
ch_thread, 566
cumulative
time_measurement_t, 606
currcore
System Management, 114
current
ch_ready_list, 554
Customer Information, 24
CH_CUSTOMER_ID_CODE, 25
CH_CUSTOMER_ID_STRING, 24
CH_CUSTOMER_LICENSE_EOS_DATE, 25
CH_CUSTOMER_LICENSE_VERSION_DATE, 25
CH_CUSTOMER_LICENSE_VERSION_MONTH, 25
CH_CUSTOMER_LICENSE_VERSION_YEAR, 25
CH_LICENSE, 25
dbg
ch_os_instance, 548
Debug, 324
default_heap
Memory Heaps, 436
Delegate Threads, 391
__ch_delegate_fn0, 392
__ch_delegate_fn1, 393
__ch_delegate_fn2, 393
__ch_delegate_fn3, 393
__ch_delegate_fn4, 395
delegateCallDirect0, 397
delegateCallDirect1, 398
delegateCallDirect2, 399
chDelegateCallDirect3, 400
chDelegateCallDirect4, 401
chDelegateCallVeneer, 395
chDelegateDispatch, 396
chDelegateDispatchTimeout, 396
delegate_fn0_t, 392
delegate_fn1_t, 392
delegate_fn2_t, 392
delegate_fn3_t, 392
delegate_fn4_t, 392
delegate_veneer_t, 392
delegate_fn0_t
Delegate Threads, 392
delegate_fn1_t
Delegate Threads, 392
delegate_fn2_t
Delegate Threads, 392
delegate_fn3_t
Delegate Threads, 392
delegate_fn4_t
Delegate Threads, 392
delegate_veneer_t
Delegate Threads, 392
delta
ch_delta_list, 517
dlist
ch_virtual_timer, 573
ch_virtual_timers_list, 575
dptr
ch_oc_object, 545
dyn_buffer_t
Dynamic Objects Factory, 495
dyn_element_t
Dynamic Objects Factory, 495
dyn_list_t
Dynamic Objects Factory, 495
dyn_mailbox_t
Dynamic Objects Factory, 496
dyn_objects_fifo_t
Dynamic Objects Factory, 496
dyn_pipe_t
Dynamic Objects Factory, 496
dyn_semaphore_t
Dynamic Objects Factory, 495
Dynamic Objects Factory, 490
__factory_init, 496
CH_CFG_FACTORY_GENERIC_BUFFERS, 493
CH_CFG_FACTORY_MAILBOXES, 494
CH_CFG_FACTORY_MAX_NAMES_LENGTH, 493
CH_CFG_FACTORY_OBJ_FIFO, 494
CH_CFG_FACTORY_OBJECTS_REGISTERY, 493
CH_CFG_FACTORY_PIPE, 494, 495
CH_CFG_FACTORY_SEMAPHORES, 493
ch_factory, 513
chFactoryCreateBuffer, 499
chFactoryCreateMailbox, 503
chFactoryCreateObjectsFIFO, 504
chFactoryCreatePipe, 506
chFactoryCreateSemaphore, 501
chFactoryDuplicateReference, 509
chFactoryFindBuffer, 500
chFactoryFindMailbox, 503
chFactoryFindObject, 498
chFactoryFindObjectByPointer, 498
chFactoryFindObjectFIFO, 505
chFactoryFindPipe, 507
chFactoryFindSemaphore, 502
chFactoryGetBuffer, 510
chFactoryGetBufferSize, 510
chFactoryGetMailbox, 511
chFactoryGetObject, 509
chFactoryGetObjectFIFO, 512
chFactoryGetPipe, 512
chFactoryGetSemaphore, 511
chFactoryRegisterObject, 497
chFactoryReleaseBuffer, 501
chFactoryReleaseMailbox, 504
chFactoryReleaseObject, 499
chFactoryReleaseObjectsFIFO, 506
chFactoryReleasePipe, 507
chFactoryReleaseSemaphore, 502
dyn_buffer_t, 495
dyn_element_t, 495
dyn_list_t, 495
dyn_mailbox_t, 496
dyn_objects_fifo_t, 496
dyn_pipe_t, 496
dyn_semaphore_t, 495
objects_factory_t, 496
registered_object_t, 495
Dynamic Threads, 311
chThdCreateFromHeap, 311
chThdCreateFromMemoryPool, 312

element
ch_dyn_mailbox, 520
ch_dyn_object, 522
ch_dyn_objects_fifo, 523
ch_dyn_pipe, 525
ch_dyn_semaphore, 526
ch_registered_static_object, 555

epending
ch_thread, 570

Event Flags, 273
__EVENTSOURCE_DATA, 275
ALL EVENTS, 275
chEvtAddEvents, 281
chEvtAddEventsI, 298
chEvtBroadcast, 297
chEvtBroadcastFlags, 287
chEvtBroadcastFlagsI, 286
chEvtBroadcastI, 297
chEvtDispatch, 288
chEvtGetAndClearEvents, 280
chEvtGetAndClearEventsI, 279
chEvtGetAndClearFlags, 283
chEvtGetAndClearFlagsI, 282
chEvtGetEventsX, 298
chEvtIsListeningI, 296
chEvtObjectInit, 294
chEvtRegister, 295
chEvtRegisterMask, 295
chEvtRegisterMaskWithFlags, 277
chEvtRegisterMaskWithFlagsI, 276
chEvtSignal, 285
chEvtSignall, 284
chEvtUnregister, 278
chEvtWaitAll, 290
chEvtWaitAllTimeout, 293
chEvtWaitAny, 289
chEvtWaitAnyTimeout, 292
chEvtWaitOne, 288
chEvtWaitOneTimeout, 291
EVENT_MASK, 275
event_source_t, 276
EVENTSOURCE_DECL, 276
evhandler_t, 276

event_listener, 581
  events, 582
  flags, 582
  listener, 582
  next, 582
  wflags, 583
EVENT_EVENT
  Event Flags, 275
event_source, 583
  next, 584
event_source_t
  Event Flags, 276
eventflags_t
  OS Types and Structures, 63
eventid_t
  OS Types and Structures, 62
eventmask_t
  OS Types and Structures, 63
events
  event_listener, 582
EVENTSOURCE_DECL
  Event Flags, 276
evhandler_t
  Event Flags, 276
ewmask
  ch_thread, 569
exitcode
  ch_thread, 568

FALSE
  Version Numbers and Identification, 30
fifo
  ch_dyn_objects_fifo, 523
fifo_list
  ch_objects_factory, 537
firstprio
  Scheduler, 97
flags
  ch_thread, 567
  event_listener, 582
free
  ch_jobs_queue, 528
  ch_objects_fifo, 539
func
  ch_virtual_timer, 573
funcp
  thread_descriptor_t, 604
guarded_memory_pool_t
  pool, 586
  sem, 586
GUARDEDMEMORYPOOL_DECL
  Memory Pools, 440
halt
  trace_event_t, 614
hash_get_s
  Objects Caches, 482
hash_next
  ch_oc_hash_header, 540
  ch_oc_lru_header, 542
  ch_oc_object, 544
hash_prev
  ch_oc_hash_header, 541
  ch_oc_lru_header, 542
  ch_oc_object, 544
hashn
  ch_objects_cache, 533
hashp
  ch_objects_cache, 533
hdr
  ch_thread, 565
header
  memory_heap, 595
heap
  heap_header, 587
heap_header
  heap, 587
  next, 587
  pages, 587
  size, 587
heap_header_t
  Memory Heaps, 432
HIGHPRIO
  Scheduler, 93
identifier
  chdebug_t, 577
IDLEPRIO
  Scheduler, 92
idlethread_base
  ch_os_instance_config, 550
idlethread_end
  ch_os_instance_config, 550
instance
  thread_descriptor_t, 604
instances
mailbox_t, 590
  buffer, 591
  cnt, 592
  qr, 592
  qw, 592
  rdptr, 591
  reset, 592
  top, 592
  wrptr, 591
Mailboxes, 365
  __MAILBOX_DATA, 366
  chMBFetchI, 378
  chMBFetchTimeout, 376
  chMBFetchTimeoutS, 377
  chMBGetFreeCountI, 380
  chMBGetSizeI, 379
  chMBGetUsedCountI, 380
  chMOBJECTInit, 367
  chMBPeekI, 381
  chMBPostAheadI, 375
  chMBPostAheadTimeout, 373
  chMBPostAheadTimeoutS, 374
  chMBPostI, 372
  chMBPostTimeout, 370
  chMBPostTimeoutS, 371
  chMBReset, 368
  chMBResetI, 369
  chMBResumeX, 382
  MAILBOX_DECL, 367
mainthread
  ch_os_instance, 548
mainthread_base
  ch_os_instance_config, 549
mainthread_end
  ch_os_instance_config, 550
mask
  ch_rfcu, 557
mbx
  ch_dyn_mailbox, 521
  ch_jobs_queue, 529
  ch_objects_fifo, 539
mbx_list
  ch_objects_factory, 537
MEM_ALIGN_MASK
Memory Alignment, 136
MEM_ALIGN_NEXT
Memory Alignment, 137
MEM_ALIGN_PREV
Memory Alignment, 136
MEM_IS_ALIGNED
Memory Alignment, 137
MEM_IS_VALID_ALIGNMENT
Memory Alignment, 137
memcore_t, 593
  basemem, 593
  topmem, 593
memgetfunc2_t
  Core Memory Manager, 421
memgetfunc_t
  Core Memory Manager, 421
Memory Alignment, 136
  MEM_ALIGN_MASK, 136
  MEM_ALIGN_NEXT, 137
  MEM_ALIGN_PREV, 136
  MEM_IS_ALIGNED, 137
  MEM_IS_VALID_ALIGNMENT, 137
Memory Heaps, 430
  __heap_init, 432
    CH_HEAP_ALIGNMENT, 431
    CH_HEAP_AREA, 431
    chHeapAlloc, 434
    chHeapAllocAligned, 433
    chHeapFree, 433
    chHeapGetSize, 435
    chHeapObjectInit, 432
    chHeapStatus, 434
    default_heap, 436
    heap_header_t, 432
    memory_heap_t, 432
Memory Management, 418
Memory Pools, 437
  __GUARDEDMEMORYPOOL_DATA, 439
  __MEMORYPOOL_DATA, 438
  chGuardedPoolAdd, 458
  chGuardedPoolAddI, 459
  chGuardedPoolAddS, 460
  chGuardedPoolAlloc, 455
  chGuardedPoolAllocTimeout, 449
  chGuardedPoolAllocTimeoutS, 448
  chGuardedPoolFree, 450
  chGuardedPoolFreeI, 456
  chGuardedPoolFreeS, 457
  chGuardedPoolGetCounterI, 454
  chGuardedPoolLoadArray, 447
  chGuardedPoolObjectInit, 454
  chGuardedPoolObjectInitAligned, 446
  chPoolAdd, 452
  chPoolAddI, 453
  chPoolAlloc, 443
  chPoolAllocI, 442
  chPoolFree, 445
  chPoolFreeI, 444
  chPoolLoadArray, 441
  chPoolObjectInit, 451
  chPoolObjectInitAligned, 440
  GUARDEDMEMORYPOOL_DECL, 440
  MEMORYPOOL_DECL, 439
memory_heap, 594
  header, 595
  mtx, 595
  provider, 595
memory_heap_t
  Memory Heaps, 432
memory_pool_t, 596
  align, 597
  next, 596
object_size, 597
provider, 597
MEMORYPOOL_DECL
Memory Pools, 439
mpool
ch_thread, 571
MS2RTC
System Management, 117
msg
trace_event_t, 613
MSG_JOB_NULL
Jobs Queues, 404
MSG_OK
Scheduler, 92
MSG_RESET
Scheduler, 92
msg_t
OS Types and Structures, 62
MSG_TIMEOUT
Scheduler, 92
msgqueue
ch_thread, 570
mtx
ch_objects_factory, 536
memory_heap, 595
mtxlist
ch_thread, 570
MUTEX_DECL
Mutexes, 249
mutex_t
Mutexes, 249
Mutexes, 246
__MUTEX_DATA, 247
chMtxGetNextMutexX, 260
chMtxGetOwnerI, 259
chMtxLock, 250
chMtxLockS, 251
chMtxObjectInit, 249
chMtxQueueNotEmptyS, 258
chMtxTryLock, 252
chMtxTryLockS, 253
chMtxUnlock, 254
chMtxUnlockAll, 257
chMtxUnlockAllIS, 256
chMtxUnlockS, 255
MUTEX_DECL, 249
mutex_t, 249
n
time_measurement_t, 606
n_ctxswc
kernel_stats_t, 589
n_irq
kernel_stats_t, 589
name
ch_os_instance_config, 549
ch_thread, 566
thread_descriptor_t, 603
trace_event_t, 613
next
ch_delta_list, 517
ch_dyn_element, 518
ch_list, 529
ch_mutex, 531
ch_priority_queue, 551
ch_queue, 552
event_listener, 582
event_source, 584
heap_header, 587
memory_pool_t, 596
pool_header, 602
NORAS
Scheduler, 92
NORMALPRIO
Scheduler, 93
ntp
trace_event_t, 612
obj_flags
ch_oc_object, 545
obj_group
ch_oc_object, 544
obj_key
ch_oc_object, 545
obj_list
ch_objects_factory, 536
obj_pool
ch_objects_factory, 536
obj_sem
ch_objects_factory, 536
obj_sem
ch_oc_object, 545
object_size
memory_pool_t, 597
_objects Caches, 480
chCacheGetObject, 485
chCacheObjectInit, 484
chCacheReadObject, 486
chCacheReleaseObject, 488
chCacheReleaseObjectI, 486
chCacheWriteObject, 487
hash_get_s, 482
lru_get_last_s, 483
objects_cache_t, 482
oc_flags_t, 481
oc_hash_header_t, 481
oc_lru_header_t, 481
oc_object_t, 481
oc_readf_t, 482
oc_writef_t, 482
_objects FIFOs, 463
chFifoObjectInit, 465
chFifoObjectInitAligned, 464
chFifoReceiveObject, 476
chFifoReceiveObjectTimeout, 478
chFifoReceiveObjectTimeoutS, 477
chFifoReturnObject, 470
chFifoReturnObjectI, 469
chFifoReturnObjectS, 469
chFifoSendObject, 473

ChibiOS/RT
chFifoSendObjectAhead, 476
chFifoSendObjectAheadI, 474
chFifoSendObjectAheadS, 475
chFifoSendObjectI, 471
chFifoSendObjectS, 472
chFifoTakeObject, 466
chFifoTakeObjectTimeout, 468
chFifoTakeObjectTimeoutS, 467
objects_fifo_t, 464
objects_cache_t
Objects Caches, 482
objects_factory_t
Dynamic Objects Factory, 496
objects_fifo_t
Objects FIFOs, 464
objn
ch_objects_cache, 533
objp
ch_registered_static_object, 555
objsz
ch_objects_cache, 534
objtyp
ch_objects_cache, 534
oc_flags_t
Objects Caches, 481
oc_hash_header_t
Objects Caches, 481
oc_lru_header_t
Objects Caches, 481
oc_object_t
Objects Caches, 481
oc_readf_t
Objects Caches, 482
oc_writef_t
Objects Caches, 482
off_ctx
chdebug_t, 578
off_flags
chdebug_t, 579
off_name
chdebug_t, 579
off_newer
chdebug.t, 579
off_older
chdebug,t, 579
off_preempt
chdebug.t, 580
off_prio
chdebug.t, 578
off_ref
chdebug.t, 580
off_state
chdebug.t, 579
off_stlimt
chdebug.t, 579
off_time
chdebug.t, 580
offset

ChibiOS/RT
provider
memory_heap, 595
memory_pool_t, 597

ptr
trace_buffer_t, 609
ptrsize
chdebug_t, 578

qr
mailbox_t, 592
queue
ch_mutex, 531
ch_registry, 556
ch_semaphore, 559
ch_thread, 565
ch_threads_queue, 572
condition_variable, 581

qw
mailbox_t, 592

rdptr
mailbox_t, 591
pipe_t, 600
rdy
trace_event_t, 613
rdymsg
ch_thread, 567
readf
ch_objects_cache, 534
ready_list_t
OS Types and Structures, 65
realprio
ch_thread, 570
reason
trace_event_t, 614
refs
ch_dyn_element, 518
ch_thread, 567
REG_HEADER
Registry, 315
REG_INSERT
Registry, 316
REG_REMOVE
Registry, 315
registered_object_t
Dynamic Objects Factory, 495
Registry, 314
__reg_object_init, 321
chRegFindThreadByName, 318
chRegFindThreadByPointer, 319
chRegFindThreadByWorkingArea, 320
chRegFirstThread, 316
chRegGetThreadNameX, 322
chRegNextThread, 317
chRegSetThreadName, 321
chRegSetThreadNameX, 322
REG_HEADER, 315
REG_INSERT, 316
REG_REMOVE, 315
registry_t
OS Types and Structures, 65
reglist
ch_os_instance, 547
ch_system, 561
Release and Licensing, 21
Release Information, 22
__CHIBIOS__, 22
CH_VERSION, 22
CH_VERSION_DATE, 23
CH_VERSION_MONTH, 23
CH_VERSION_NICKNAME, 23
CH_VERSION_PATCH, 23
CH_VERSION_STABLE, 22
CH_VERSION_YEAR, 23
reload
ch_virtual_timer, 573
reset
mailbox_t, 592
pipe_t, 600
Restrictions, 55
rfcu
ch_os_instance, 547
ch_system, 561
rfcu_mask_t
Runtime Faults Collection Unit, 72
rfcu_t
Runtime Faults Collection Unit, 72
rlist
ch_os_instance, 547
rmtx
pipe_t, 601
rqueue
ch_thread, 566
RT Kernel, 28
RTC2MS
System Management, 119
RTC2S
System Management, 118
RTC2US
System Management, 119
rtcnt_t
OS Types and Structures, 61
rtr
pipe_t, 600
rtstamp
trace_event_t, 612
rttime_t
OS Types and Structures, 61
Runtime Faults Collection Unit, 71
__rfcu_object_init, 73
CH_RFCU_ALL_FAULTS, 71
chRFCUCollectFaultsI, 72
chRFCUGetAndClearFaultsI, 72
rfcu_mask_t, 72
rfcu_t, 72
S2RTC
System Management, 116

ChibiOS/RT
INDEX

THD_FUNCTION, 185
THD_WORKING_AREA, 184
THD_WORKING_AREA_BASE, 184
THD_WORKING_AREA_END, 185
THD_WORKING_AREA_SIZE, 183
THREADS_QUEUEDECL, 183

THREADS_QUEUEDECL
Threads, 183
threads_queue_t
OS Types and Structures, 65
threadsize
chdebug_t, 578
ticks
ch_thread, 567
time
ch_thread, 567
trace_event_t, 612

Time and Intervals, 139
chTimeAddX, 149
chTimeDiffX, 150
chTimeI2MS, 148
chTimeI2S, 148
chTimeI2US, 149
chTimesInRangeX, 150
chTimeMS2I, 147
chTimeS2I, 146
chTimeStampAddX, 151
chTimeStampDiffX, 151
chTimeStampsInRangeX, 152
chTimeUS2I, 147
sysinterval_t, 145
sysfirst_t, 145
systimestamp_t, 145
time_conv_t, 146
time_measurement_t, 604

TIME_I2S
Time and Intervals, 143
TIME_I2US
Time and Intervals, 144
TIME_IMMEDIATE
Time and Intervals, 140
TIME_INFINITE
Time and Intervals, 140
TIME_MAX_INTERVAL
Time and Intervals, 141
TIME_MAX_SYSTIME
Time and Intervals, 141
time_measurement_t, 604
    best, 605
cumulative, 606
    last, 605
    n, 606
worst, 605
TIME_MS2I
Time and Intervals, 142
time_msecs_t
Time and Intervals, 146
TIME_S2I
Time and Intervals, 141
time_secs_t
Time and Intervals, 145
TIME_US2I
Time and Intervals, 142
time usecs t
Time and Intervals, 146
timesize
chdebug t, 578

TM_CALIBRATION_LOOP
Time Measurement, 223
tm_calibration_t, 606
    offset, 607
tmc
ch_system, 561
tmode_t
OS Types and Structures, 62
top
mailbox_t, 591
pipe_t, 599
topmem
memcore_t, 593
tp
trace_event_t, 613

TSPRO T
OS Types and Structures, 62
trace_buffer
ch_os_instance, 548
trace_buffer t, 607
buffer, 609
ptr, 609
size, 609
suspended, 609
trace_event_t, 610
halt, 614

ChibiOS/RT
isr, 613
msg, 613
name, 613
ntp, 612
rdy, 613
reason, 614
rtstamp, 612
state, 612
sw, 613
time, 612
tp, 613
type, 612
up1, 614
up2, 614
user, 614
wtobjp, 612
trace_next
Tracing, 337

__trace_halt, 339
__trace_isr_enter, 338
__trace_isr_leave, 339
__trace_object_init, 337
__trace_ready, 337
__trace_switch, 338
CH_DBG_TRACE_BUFFER_SIZE, 336
CH_DBG_TRACE_MASK, 336
chTraceResume, 343
chTraceResumeI, 343
chTraceSuspend, 342
chTraceSuspendI, 341
chTraceWrite, 340
chTraceWritel, 339
trace_next, 337
trefs_t
OS Types and Structures, 62
TRUE
Version Numbers and Identification, 30
tslices_t
OS Types and Structures, 62
tstate_t
OS Types and Structures, 62
type
trace_event_t, 612
u
ch_thread, 569
ucnt_t
OS Types and Structures, 63
unlikely
OS Types and Structures, 61
up1
trace_event_t, 614
up2
trace_event_t, 614
US2RTC
System Management, 118
user
trace_event_t, 614

version
chdebug_t, 578
Version Numbers and Identification, 29, 350
__CHIBIOS_OSLIB__, 350
__CHIBIOS_RT__, 29
__oslib_init, 351
CH_KERNEL_MAJOR, 30
CH_KERNEL_MINOR, 30
CH_KERNEL_PATCH, 30
CH_KERNEL_STABLE, 29
CH_KERNEL_VERSION, 30
CH_OSLIB_MAJOR, 351
CH_OSLIB_MINOR, 351
CH_OSLIB_PATCH, 351
CH_OSLIB_STABLE, 350
CH_OSLIB_VERSION, 350
FALSE, 30
TRUE, 30

Virtual Timers, 153
__vt_object_init, 178
chVTDoReset, 157
chVTDoResetContinuousI, 156
chVTDoSet, 155
chVTDoTickI, 159
chVTGetReloadIntervalX, 175
chVTGetRemainingIntervalI, 158
chVTGetSystemTime, 162
chVTGetSystemTimeX, 162
chVTGetTimersStatusI, 166
chVTGetTimeStamp, 176
chVTGetTimeStampI, 160
chVTIsArmed, 168
chVTIsArmedI, 167
chVTIsSystemTimeWithin, 165
chVTIsSystemTimeWithinI, 164
chVObjectInit, 161
chVTReset, 170
chVTResetI, 169
chVTResetTimeStampl, 161
chVTSet, 172
chVTSetContinuous, 174
chVTSetContinuousI, 173
chVTSetI, 171
chVTSetReloadIntervalX, 175
chVTTimeElapsedSinceX, 163
vt_enqueue, 154
vt_insert_first, 154
virtual_timer_t
OS Types and Structures, 64
virtual_timers_list_t
OS Types and Structures, 64
vt_enqueue
Virtual Timers, 154
vt_insert_first
Virtual Timers, 154
vtfunc_t
OS Types and Structures, 64
vtlist
  ch_os_instance, 547

wabase
  ch_thread, 566
waiting
  ch_thread, 570
wbase
  thread_descriptor_t, 603
wend
  thread_descriptor_t, 603
wflags
  event_listener, 583
wmtx
  pipe_t, 601
worst
  time_measurement_t, 605
writef
  ch_objects_cache, 535
wrptr
  mailbox_t, 591
  pipe_t, 599
wtmtx
  ch_thread, 569
wtobjp
  ch_thread, 568
  trace_event_t, 612
wtr
  pipe_t, 600
wtsemp
  ch_thread, 569
wttrp
  ch_thread, 568

zero
  chdebug_t, 577