# Contents

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

2 Kernel Concepts .................................... 3
   2.1 Naming Conventions ................................ 3
   2.2 API Name Suffixes .................................. 3
   2.3 Interrupt Classes ................................... 4
   2.4 System States ....................................... 4
   2.5 Scheduling .......................................... 6
   2.6 Thread States ....................................... 7
   2.7 Priority Levels ..................................... 7
   2.8 Thread Working Area ................................ 7

3 Module Index ........................................... 9
   3.1 Modules ............................................. 9

4 Hierarchical Index ................................... 11
   4.1 Class Hierarchy ..................................... 11

5 Data Structure Index .................................. 13
   5.1 Data Structures ..................................... 13

6 File Index ............................................. 15
   6.1 File List ............................................ 15

7 Module Documentation ................................ 19
   7.1 Release and Licensing .............................. 19
      7.1.1 Detailed Description .......................... 19
   7.2 Release Information ................................. 20
      7.2.1 Detailed Description .......................... 20
      7.2.2 Macro Definition Documentation .............. 20
         7.2.2.1 _CHIBIOS_ ................................. 20
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.2.2.2</td>
<td>CH_VERSION_STABLE</td>
<td>20</td>
</tr>
<tr>
<td>7.2.2.3</td>
<td>CH_VERSION</td>
<td>20</td>
</tr>
<tr>
<td>7.2.2.4</td>
<td>CH_VERSION_YEAR</td>
<td>20</td>
</tr>
<tr>
<td>7.2.2.5</td>
<td>CH_VERSION_MONTH</td>
<td>20</td>
</tr>
<tr>
<td>7.2.2.6</td>
<td>CH_VERSION_PATCH</td>
<td>21</td>
</tr>
<tr>
<td>7.2.2.7</td>
<td>CH_VERSION_NICKNAME</td>
<td>21</td>
</tr>
<tr>
<td>7.3</td>
<td>Customer Information</td>
<td>22</td>
</tr>
<tr>
<td>7.3.1</td>
<td>Detailed Description</td>
<td>22</td>
</tr>
<tr>
<td>7.3.2</td>
<td>Macro Definition Documentation</td>
<td>22</td>
</tr>
<tr>
<td>7.3.2.1</td>
<td>CH_CUSTOMER_ID_STRING</td>
<td>22</td>
</tr>
<tr>
<td>7.3.2.2</td>
<td>CH_CUSTOMER_ID_CODE</td>
<td>22</td>
</tr>
<tr>
<td>7.3.2.3</td>
<td>CH_LICENSE</td>
<td>22</td>
</tr>
<tr>
<td>7.4</td>
<td>License Settings</td>
<td>23</td>
</tr>
<tr>
<td>7.4.1</td>
<td>Detailed Description</td>
<td>23</td>
</tr>
<tr>
<td>7.4.2</td>
<td>Macro Definition Documentation</td>
<td>23</td>
</tr>
<tr>
<td>7.4.2.1</td>
<td>CH_LICENSE_TYPE_STRING</td>
<td>23</td>
</tr>
<tr>
<td>7.4.2.2</td>
<td>CH LICENSE_ID_STRING</td>
<td>24</td>
</tr>
<tr>
<td>7.4.2.3</td>
<td>CH LICENSE_ID_CODE</td>
<td>24</td>
</tr>
<tr>
<td>7.4.2.4</td>
<td>CH_LICENSE_MODIFIABLE_CODE</td>
<td>24</td>
</tr>
<tr>
<td>7.4.2.5</td>
<td>CH_LICENSE_FEATURES</td>
<td>24</td>
</tr>
<tr>
<td>7.4.2.6</td>
<td>CH_LICENSE_MAX_DEPLOY</td>
<td>24</td>
</tr>
<tr>
<td>7.5</td>
<td>RT Kernel</td>
<td>25</td>
</tr>
<tr>
<td>7.5.1</td>
<td>Detailed Description</td>
<td>25</td>
</tr>
<tr>
<td>7.6</td>
<td>Version Numbers and Identification</td>
<td>26</td>
</tr>
<tr>
<td>7.6.1</td>
<td>Detailed Description</td>
<td>26</td>
</tr>
<tr>
<td>7.6.2</td>
<td>Macro Definition Documentation</td>
<td>26</td>
</tr>
<tr>
<td>7.6.2.1</td>
<td><em>CHIBIOS_RT</em></td>
<td>26</td>
</tr>
<tr>
<td>7.6.2.2</td>
<td>CH_KERNEL_STABLE</td>
<td>26</td>
</tr>
<tr>
<td>7.6.2.3</td>
<td>CH_KERNEL_VERSION</td>
<td>27</td>
</tr>
<tr>
<td>7.6.2.4</td>
<td>CH_KERNEL_MAJOR</td>
<td>27</td>
</tr>
<tr>
<td>7.6.2.5</td>
<td>CH_KERNEL_MINOR</td>
<td>27</td>
</tr>
<tr>
<td>7.6.2.6</td>
<td>CH_KERNEL_PATCH</td>
<td>27</td>
</tr>
<tr>
<td>7.6.2.7</td>
<td>FALSE</td>
<td>27</td>
</tr>
<tr>
<td>7.6.2.8</td>
<td>TRUE</td>
<td>27</td>
</tr>
<tr>
<td>7.6.3</td>
<td>Function Documentation</td>
<td>27</td>
</tr>
<tr>
<td>7.6.3.1</td>
<td>chSysHalt(const char *reason)</td>
<td>27</td>
</tr>
<tr>
<td>7.7</td>
<td>Configuration</td>
<td>29</td>
</tr>
<tr>
<td>7.7.1</td>
<td>Detailed Description</td>
<td>29</td>
</tr>
<tr>
<td>7.8</td>
<td>Options</td>
<td>30</td>
</tr>
<tr>
<td>7.8.1</td>
<td>Detailed Description</td>
<td>30</td>
</tr>
<tr>
<td>7.8.2</td>
<td>Macro Definition Documentation</td>
<td>33</td>
</tr>
<tr>
<td>-------</td>
<td>--------------------------------</td>
<td>----</td>
</tr>
<tr>
<td>7.8.2.1</td>
<td>CH_CFG_ST_RESOLUTION</td>
<td>33</td>
</tr>
<tr>
<td>7.8.2.2</td>
<td>CH_CFG_ST_FREQUENCY</td>
<td>33</td>
</tr>
<tr>
<td>7.8.2.3</td>
<td>CH_CFG_INTERVALS_SIZE</td>
<td>33</td>
</tr>
<tr>
<td>7.8.2.4</td>
<td>CH_CFG_TIME_TYPES_SIZE</td>
<td>33</td>
</tr>
<tr>
<td>7.8.2.5</td>
<td>CH_CFG_ST_TIMEDI TELA</td>
<td>33</td>
</tr>
<tr>
<td>7.8.2.6</td>
<td>CH_CFG_TIME_QUANTUM</td>
<td>33</td>
</tr>
<tr>
<td>7.8.2.7</td>
<td>CH_CFG_MEMCORE_SIZE</td>
<td>34</td>
</tr>
<tr>
<td>7.8.2.8</td>
<td>CH_CFG_NO_IDLE_THREAD</td>
<td>34</td>
</tr>
<tr>
<td>7.8.2.9</td>
<td>CH_CFG_OPTIMIZE_SPEED</td>
<td>34</td>
</tr>
<tr>
<td>7.8.2.10</td>
<td>CH_CFG_USE_TM</td>
<td>34</td>
</tr>
<tr>
<td>7.8.2.11</td>
<td>CH_CFG_USE_REGISTRY</td>
<td>34</td>
</tr>
<tr>
<td>7.8.2.12</td>
<td>CH_CFG_USE_WAITEXIT</td>
<td>34</td>
</tr>
<tr>
<td>7.8.2.13</td>
<td>CH_CFG_USE_SEMAPHORES</td>
<td>35</td>
</tr>
<tr>
<td>7.8.2.14</td>
<td>CH_CFG_USE_SEMAPHORES_PRIORITY</td>
<td>35</td>
</tr>
<tr>
<td>7.8.2.15</td>
<td>CH_CFG_USE_MUTEXES</td>
<td>35</td>
</tr>
<tr>
<td>7.8.2.16</td>
<td>CH_CFG_USE_MUTEXES_RECURSIVE</td>
<td>35</td>
</tr>
<tr>
<td>7.8.2.17</td>
<td>CH_CFG_USE_CONDVARS</td>
<td>35</td>
</tr>
<tr>
<td>7.8.2.18</td>
<td>CH_CFG_USE_CONDVARS_TIMEOUT</td>
<td>36</td>
</tr>
<tr>
<td>7.8.2.19</td>
<td>CH_CFG_USE_EVENTS</td>
<td>36</td>
</tr>
<tr>
<td>7.8.2.20</td>
<td>CH_CFG_USE_EVENTS_TIMEOUT</td>
<td>36</td>
</tr>
<tr>
<td>7.8.2.21</td>
<td>CH_CFG_USE_MESSAGES</td>
<td>36</td>
</tr>
<tr>
<td>7.8.2.22</td>
<td>CH_CFG_USE_MESSAGES_PRIORITY</td>
<td>36</td>
</tr>
<tr>
<td>7.8.2.23</td>
<td>CH_CFG_USE_MAILBOXES</td>
<td>37</td>
</tr>
<tr>
<td>7.8.2.24</td>
<td>CH_CFG_USE_MEMCORE</td>
<td>37</td>
</tr>
<tr>
<td>7.8.2.25</td>
<td>CH_CFG_USE_HEAP</td>
<td>37</td>
</tr>
<tr>
<td>7.8.2.26</td>
<td>CH_CFG_USE_MEMPOOLS</td>
<td>37</td>
</tr>
<tr>
<td>7.8.2.27</td>
<td>CH_CFG_USE_OBJ_FIFOS</td>
<td>37</td>
</tr>
<tr>
<td>7.8.2.28</td>
<td>CH_CFG_USE_PIPES</td>
<td>38</td>
</tr>
<tr>
<td>7.8.2.29</td>
<td>CH_CFG_USE_DYNAMIC</td>
<td>38</td>
</tr>
<tr>
<td>7.8.2.30</td>
<td>CH_CFG_USE_FACTORY</td>
<td>38</td>
</tr>
<tr>
<td>7.8.2.31</td>
<td>CH_CFG_FACTORY_MAX_NAMES_LENGTH</td>
<td>38</td>
</tr>
<tr>
<td>7.8.2.32</td>
<td>CH_CFG_FACTORY_OBJECTS_REGISTRY</td>
<td>38</td>
</tr>
<tr>
<td>7.8.2.33</td>
<td>CH_CFG_FACTORY_GENERIC_BUFFERS</td>
<td>38</td>
</tr>
<tr>
<td>7.8.2.34</td>
<td>CH_CFG_FACTORY_SEMAPHORES</td>
<td>38</td>
</tr>
<tr>
<td>7.8.2.35</td>
<td>CH_CFG_FACTORY_MAILBOXES</td>
<td>38</td>
</tr>
<tr>
<td>7.8.2.36</td>
<td>CH_CFG_FACTORY_OBJ_FIFOS</td>
<td>39</td>
</tr>
<tr>
<td>7.8.2.37</td>
<td>CH_CFG_FACTORY_PIPES</td>
<td>39</td>
</tr>
<tr>
<td>7.8.2.38</td>
<td>CH_DBG_STATISTICS</td>
<td>39</td>
</tr>
<tr>
<td>7.8.2.39</td>
<td>CH_DBG_SYSTEM_STATE_CHECK</td>
<td>39</td>
</tr>
<tr>
<td>Section</td>
<td>Name</td>
<td>Page</td>
</tr>
<tr>
<td>---------</td>
<td>------</td>
<td>------</td>
</tr>
<tr>
<td>7.8.2.40</td>
<td>CH_DBG_ENABLE_CHECKS</td>
<td>39</td>
</tr>
<tr>
<td>7.8.2.41</td>
<td>CH_DBG_ENABLE_ASSERTS</td>
<td>39</td>
</tr>
<tr>
<td>7.8.2.42</td>
<td>CH_DBG_TRACE_MASK</td>
<td>39</td>
</tr>
<tr>
<td>7.8.2.43</td>
<td>CH_DBG_TRACE_BUFFER_SIZE</td>
<td>40</td>
</tr>
<tr>
<td>7.8.2.44</td>
<td>CH_DBG_ENABLE_STACK_CHECK</td>
<td>40</td>
</tr>
<tr>
<td>7.8.2.45</td>
<td>CH_DBG_FILL_THREADS</td>
<td>40</td>
</tr>
<tr>
<td>7.8.2.46</td>
<td>CH_DBG_THREADS_PROFILING</td>
<td>40</td>
</tr>
<tr>
<td>7.8.2.47</td>
<td>CH_CFG_SYSTEM_EXTRA_FIELDS</td>
<td>40</td>
</tr>
<tr>
<td>7.8.2.48</td>
<td>CH_CFG_SYSTEM_INIT_HOOK</td>
<td>40</td>
</tr>
<tr>
<td>7.8.2.49</td>
<td>CH_CFG_THREAD_EXTRA_FIELDS</td>
<td>41</td>
</tr>
<tr>
<td>7.8.2.50</td>
<td>CH_CFG_THREAD_INIT_HOOK</td>
<td>41</td>
</tr>
<tr>
<td>7.8.2.51</td>
<td>CH_CFG_THREAD_EXIT_HOOK</td>
<td>41</td>
</tr>
<tr>
<td>7.8.2.52</td>
<td>CH_CFG_CONTEXT_SWITCH_HOOK</td>
<td>41</td>
</tr>
<tr>
<td>7.8.2.53</td>
<td>CH_CFG_IRQ_PROLOGUE_HOOK</td>
<td>41</td>
</tr>
<tr>
<td>7.8.2.54</td>
<td>CH_CFG_IRQ_EPILOGUE_HOOK</td>
<td>42</td>
</tr>
<tr>
<td>7.8.2.55</td>
<td>CH_CFG_IDLE_ENTER_HOOK</td>
<td>42</td>
</tr>
<tr>
<td>7.8.2.56</td>
<td>CH_CFG_IDLE_LEAVE_HOOK</td>
<td>42</td>
</tr>
<tr>
<td>7.8.2.57</td>
<td>CH_CFG_IDLE_LOOP_HOOK</td>
<td>42</td>
</tr>
<tr>
<td>7.8.2.58</td>
<td>CH_CFG_SYSTEM_TICK_HOOK</td>
<td>42</td>
</tr>
<tr>
<td>7.8.2.59</td>
<td>CH_CFG_SYSTEM_HALT_HOOK</td>
<td>43</td>
</tr>
<tr>
<td>7.8.2.60</td>
<td>CH_CFG_TRACE_HOOK</td>
<td>43</td>
</tr>
<tr>
<td>7.9</td>
<td>Checks</td>
<td>44</td>
</tr>
<tr>
<td>7.10</td>
<td>Restrictions</td>
<td>45</td>
</tr>
<tr>
<td>7.11</td>
<td>Base Kernel Services</td>
<td>46</td>
</tr>
<tr>
<td>7.12</td>
<td>Memory Alignment</td>
<td>47</td>
</tr>
<tr>
<td>7.13</td>
<td>System Management</td>
<td>49</td>
</tr>
</tbody>
</table>
7.14.2.5 IDLEPRIO ................................................. 72
7.14.2.6 LOWPRIO ............................................. 72
7.14.2.7 NORMALPRIO ....................................... 72
7.14.2.8 HIGHPRIO .......................................... 72
7.14.2.9 CH_STATE_READY .................................... 72
7.14.2.10 CH_STATE_CURRENT ............................... 72
7.14.2.11 CH_STATE_WTSTART .............................. 72
7.14.2.12 CH_STATE_SUSPENDED ............................ 72
7.14.2.13 CH_STATE_QUEUE .................................. 72
7.14.2.14 CH_STATE_WTSEM .................................. 72
7.14.2.15 CH_STATE_WTMTX ................................. 73
7.14.2.16 CH_STATE_WTCOND ............................... 73
7.14.2.17 CH_STATE_SLEEPING .............................. 73
7.14.2.18 CH_STATE_WTEXIT ................................ 73
7.14.2.19 CH_STATE_WTOREVT ............................. 73
7.14.2.20 CH_STATE_WTANDEVT ............................ 73
7.14.2.21 CH_STATE_SNDMSGQ .............................. 73
7.14.2.22 CH_STATE_SNDMSG ................................ 73
7.14.2.23 CH_STATE_WTMSG ................................ 73
7.14.2.24 CH_STATE_FINAL ................................. 73
7.14.2.25 CH_STATE_NAMES ................................. 73
7.14.2.26 CH_FLAG_MODE_MASK ............................ 74
7.14.2.27 CH_FLAG_MODE_STATIC ......................... 74
7.14.2.28 CH_FLAG_MODE HEAP ............................. 74
7.14.2.29 CH_FLAG_MODE_MPOOL ......................... 74
7.14.2.30 CH_FLAG_TERMINATE ......................... 74
7.14.2.31 firstprio ........................................ 74
7.14.2.32 currp ............................................ 74
7.14.2.33 ___CH_STRINGIFY ................................. 74

7.14.3 Typedef Documentation ........................................ 74
7.14.3.1 thread_t .......................................... 74
7.14.3.2 thread_reference_t ............................... 74
7.14.3.3 threads_list_t .................................. 75
7.14.3.4 threads_queue_t ................................ 75
7.14.3.5 ready_list_t ..................................... 75
7.14.3.6 vtfunc_t .......................................... 75
7.14.3.7 virtual_timer_t .................................. 75
7.14.3.8 virtual_timers_list_t ......................... 75
7.14.3.9 system_debug_t .................................. 75
7.14.3.10 ch_system_t .................................... 75
7.14.4 Function Documentation

7.14.4.1 _scheduler_init(void) .................................................. 75
7.14.4.2 queue_prio_insert(thread_t *tp, threads_queue_t *tqp) ............. 76
7.14.4.3 queue_insert(thread_t *tp, threads_queue_t *tqp) .................. 76
7.14.4.4 queue_fifo_remove(threads_queue_t *tqp) ............................ 76
7.14.4.5 queue_lifo_remove(threads_queue_t *tqp) ............................ 77
7.14.4.6 queue_dequeue(thread_t *tp) ......................................... 77
7.14.4.7 list_insert(thread_t *tp, threads_list_t *tlp) ....................... 77
7.14.4.8 list_remove(threads_list_t *tlp) .................................... 78
7.14.4.9 chSchReadyI(thread_t *tp) ........................................... 78
7.14.4.10 chSchReadyAheadI(thread_t *tp) ..................................... 79
7.14.4.11 chSchGoSleepS(tstate_t newstate) ................................... 80
7.14.4.12 chSchGoSleepTimeoutS(tstate_t newstate, sysinterval_t timeout) .. 81
7.14.4.13 chSchWakeuS(thread_t *ntp, msg_t msg) ............................ 82
7.14.4.14 chSchRescheduleS(void) .............................................. 82
7.14.4.15 chSchIsPreemptionRequired(void) ................................... 83
7.14.4.16 chSchDoRescheduleBehind(void) ................................... 83
7.14.4.17 chSchDoRescheduleAhead(void) ..................................... 84
7.14.4.18 chSchDoReschedule(void) ............................................ 84
7.14.4.19 list_init(threads_list_t *tlp) ....................................... 85
7.14.4.20 list_isempty(threads_list_t *tlp) .................................. 85
7.14.4.21 list_notempty(threads_list_t *tlp) .................................. 85
7.14.4.22 queue_init(threads_queue_t *tqp) ................................... 86
7.14.4.23 queue_isempty(const threads_queue_t *tqp) ......................... 86
7.14.4.24 queue_notempty(const threads_queue_t *tqp) ....................... 87
7.14.4.25 chSchIsRescRequiredI(void) ...................................... 87
7.14.4.26 chSchCanYieldS(void) ............................................. 88
7.14.4.27 chSchDoYieldS(void) .............................................. 88
7.14.4.28 chSchPreemption(void) ............................................. 89

7.14.5 Variable Documentation .................................................. 89
7.14.5.1 ch ................................................................. 89

7.15 Time and Intervals ......................................................... 90
7.15.1 Detailed Description ..................................................... 90
7.15.2 Macro Definition Documentation ....................................... 91
7.15.2.1 TIME_IMMEDIATE .................................................. 91
7.15.2.2 TIME_INFINITE .................................................. 91
7.15.2.3 TIME_MAX_INTERVAL ................................................ 91
7.15.2.4 TIME_MAX_SYSTIME ............................................. 91
7.15.2.5 TIME_S2I .......................................................... 91
7.15.2.6 TIME_MS2I .......................................................... 92
7.15.2.7  TIME_US2I .......................................................... 92
7.15.2.8  TIME_I2S ........................................................... 94
7.15.2.9  TIME_I2MS ......................................................... 94
7.15.2.10 TIME_I2US ......................................................... 95

7.15.3 Typedef Documentation ............................................. 95
  7.15.3.1  systime_t ....................................................... 95
  7.15.3.2  sysinterval_t .................................................. 96
  7.15.3.3  time_secs_t ................................................... 96
  7.15.3.4  time_msecs_t ................................................ 96
  7.15.3.5  time_usecs_t ............................................... 96
  7.15.3.6  time_conv_t ................................................ 96

7.15.4 Function Documentation ........................................... 96
  7.15.4.1  chTimeS2I(time_secs_t secs) .............................. 96
  7.15.4.2  chTimeMS2I(time_msecs_t msec) ......................... 97
  7.15.4.3  chTimeUS2I(time_usecs_t usec) ......................... 97
  7.15.4.4  chTimeI2S(sysinterval_t interval) .................... 98
  7.15.4.5  chTimeI2MS(sysinterval_t interval) .................. 98
  7.15.4.6  chTimeI2US(sysinterval_t interval) .................. 98
  7.15.4.7  chTimeAddX(systime_t systime, sysinterval_t interval) 99
  7.15.4.8  chTimeDiffX(systime_t start, systime_t end) ...... 99
  7.15.4.9  chTimelsInRangeX(systime_t time, systime_t start, systime_t end) 99

7.16 Virtual Timers .......................................................... 101
  7.16.1 Detailed Description ........................................... 101
  7.16.2 Function Documentation ....................................... 101
    7.16.2.1  _vt_init(void) ........................................... 101
    7.16.2.2  chVTDoSetI(virtual_timer_t *vtp, sysinterval_t delay, vtfunc_t vtfunc, void *par) 102
    7.16.2.3  chVTDoResetI(virtual_timer_t *vtp) .................. 103
    7.16.2.4  chVTObjectInit(virtual_timer_t *vtp) ............... 104
    7.16.2.5  chVTGetSystemTimeX(void) ............................ 104
    7.16.2.6  chVTGetSystemTime(void) ............................. 105
    7.16.2.7  chVTimeElapsedSinceX(systime_t systime) .......... 105
    7.16.2.8  chVTIsSystemTimeWithinX(systime_t start, systime_t end) 106
    7.16.2.9  chVTIsSystemTimeWithin(systime_t start, systime_t end) 107
    7.16.2.10 chVTGetTimersStateI(sysinterval_t *timep) ........ 108
    7.16.2.11 chVTIsArmedI(const virtual_timer_t *vtp) .......... 108
    7.16.2.12 chVTIsArmed(const virtual_timer_t *vtp) .......... 109
    7.16.2.13 chVTRestset(virtual_timer_t *vtp) ................ 110
    7.16.2.14 chVTRestset(virtual_timer_t *vtp) ................ 111
    7.16.2.15 chVTSetI(virtual_timer_t *vtp, sysinterval_t delay, vtfunc_t vtfunc, void *par) 112
    7.16.2.16 chVTSet(virtual_timer_t *vtp, sysinterval_t delay, vtfunc_t vtfunc, void *par) 113
7.16.2.17 chVTDoTickI(void) .................................................. 114

7.17 Threads ................................................................. 116

7.17.1 Detailed Description .............................................. 116

7.17.2 Macro Definition Documentation ............................... 119

7.17.2.1 _THREADS_QUEUE_DATA ........................................ 119
7.17.2.2 _THREADS_QUEUEDECL ......................................... 119
7.17.2.3 THD_WORKING_AREA_SIZE .................................... 119
7.17.2.4 THD_WORKING_AREA ............................................. 119
7.17.2.5 THD_WORKING_AREA_BASE ..................................... 120
7.17.2.6 THD_WORKING_AREA_END ....................................... 120
7.17.2.7 THD_FUNCTION .................................................. 120
7.17.2.8 chThdSleepSeconds .............................................. 120
7.17.2.9 chThdSleepMilliseconds ....................................... 121
7.17.2.10 chThdSleepMicroseconds ..................................... 121

7.17.3 Typedef Documentation ........................................... 121

7.17.3.1 tfunc_t .......................................................... 121

7.17.4 Function Documentation .......................................... 121

7.17.4.1 _thread_init(thread_t *tp, const char *name, tprio_t prio) ........................................ 121
7.17.4.2 _thread_memfill(uint8_t *startp, uint8_t *endp, uint8_t v) ........................................ 122
7.17.4.3 chThdCreateSuspendedI(const thread_descriptor_t *tdp) ......................................... 122
7.17.4.4 chThdCreateSuspended(const thread_descriptor_t *tdp) ........................................ 123
7.17.4.5 chThdCreateI(const thread_descriptor_t *tdp) .................................................. 124
7.17.4.6 chThdCreate(const thread_descriptor_t *tdp) .................................................. 125
7.17.4.7 chThdCreateStatic(void *wsp, size_t size, tprio_t prio, tfunc_t pf, void *arg) ................... 126
7.17.4.8 chThdStart(thread_t *tp) ........................................ 127
7.17.4.9 chThdAddRef(thread_t *tp) ...................................... 128
7.17.4.10 chThdRelease(thread_t *tp) ..................................... 129
7.17.4.11 chThdExit(msg_t msg) ........................................... 130
7.17.4.12 chThdExitS(msg_t msg) .......................................... 131
7.17.4.13 chThdWait(thread_t *tp) ......................................... 132
7.17.4.14 chThdSetPriority(tprio_t newprio) .............................. 133
7.17.4.15 chThdTerminate(thread_t *tp) ................................... 134
7.17.4.16 chThdSleep(sysinterval_t time) ................................ 135
7.17.4.17 chThdSleepUntil(sysinterval_t prev, sysinterval_t next) ..................................... 136
7.17.4.18 chThdSleepUntilWindowed(sysinterval_t prev, sysinterval_t next) ............................. 137
7.17.4.19 chThdYield(void) ................................................ 138
7.17.4.20 chThdSuspendS(thread_reference_t *trp) ......................... 139
7.17.4.21 chThdSuspendTimeoutS(thread_reference_t *trp, sysinterval_t timeout) ......................... 140
7.17.4.22 chThdResumeI(thread_reference_t *trp, msg_t msg) ........................................ 141
7.17.4.23 chThdResumeS(thread_reference_t *trp, msg_t msg) ........................................ 141
7.17.4.24 chThdResume(thread_reference_t *trp, msg_t msg) ........................................... 142
7.17.4.25 chThdEnqueueTimeoutS(threads_queue_t *tqp, sysinterval_t timeout) ............... 143
7.17.4.26 chThdDequeueNextI(threads_queue_t *tqp, msg_t msg) ..................................... 144
7.17.4.27 chThdDequeueAllI(threads_queue_t *tqp, msg_t msg) ....................................... 144
7.17.4.28 chThdGetSelfX(void) .......................................................................................... 145
7.17.4.29 chThdGetPriorityX(void) ...................................................................................... 145
7.17.4.30 chThdGetTicksX(thread_t *tp) ............................................................................... 146
7.17.4.31 chThdGetWorkingAreaX(thread_t *tp) .................................................................. 146
7.17.4.32 chThdTerminatedX(thread_t *tp) ......................................................................... 147
7.17.4.33 chThdShouldTerminateX(void) ............................................................................ 147
7.17.4.34 chThdStartI(thread_t *tp) .................................................................................... 147
7.17.4.35 chThdSleepS(sysinterval_t ticks) ........................................................................... 148
7.17.4.36 chThdQueueObjectInit(threads_queue_t *tqp) ....................................................... 149
7.17.4.37 chThdQueueIsEmptyI(threads_queue_t *tqp) ......................................................... 149
7.17.4.38 chThdDoDequeueNextI(threads_queue_t *tqp, msg_t msg) ................................. 150

7.18 Time Measurement ................................................................. 152
7.18.1 Detailed Description .............................................................. 152
7.18.2 Function Documentation ........................................................ 152
7.18.2.1 _tm_init(void) ................................................................. 152
7.18.2.2 chTMObjectInit(time_measurement_t *tmp) ......................................................... 153
7.18.2.3 chTMStartMeasurementX(time_measurement_t *tmp) ........................................... 153
7.18.2.4 chTMStopMeasurementX(time_measurement_t *tmp) ............................................ 153
7.18.2.5 chTMChainMeasurementToX(time_measurement_t *tmp1, time_measurement_t *tmp2) .................................................................................................................... 153

7.19 Synchronization ................................................................. 155
7.19.1 Detailed Description .............................................................. 155
7.20 Counting Semaphores ........................................................... 156
7.20.1 Detailed Description .............................................................. 156
7.20.2 Macro Definition Documentation ........................................... 157
7.20.2.1 _SEMAPHORE_DATA ......................................................................................... 157
7.20.2.2 SEMAPHORE_DECL ....................................................................................... 157
7.20.3 Typedef Documentation ........................................................ 158
7.20.3.1 semaphore_t ..................................................................................................... 158
7.20.4 Function Documentation ........................................................ 158
7.20.4.1 chSemObjectInit(semaphore_t *sp, cnt_t n) ....................................................... 158
7.20.4.2 chSemReset(semaphore_t *sp, cnt_t n) .............................................................. 158
7.20.4.3 chSemResetI(semaphore_t *sp, cnt_t n) ............................................................. 159
7.20.4.4 chSemWait(semaphore_t *sp) ............................................................................ 160
7.20.4.5 chSemWaitS(semaphore_t *sp) .......................................................................... 161
7.20.4.6 chSemWaitTimeout(semaphore_t *sp, sysinterval_t timeout) ......................... 162
7.20.4.7 chSemWaitTimeoutS(semaphore_t *sp, sysinterval_t timeout) 163
7.20.4.8 chSemSignal(semaphore_t *sp) 164
7.20.4.9 chSemSignalI(semaphore_t *sp) 165
7.20.4.10 chSemAddCounterI(semaphore_t *sp, cnt_t n) 166
7.20.4.11 chSemSignalWait(semaphore_t *sps, semaphore_t *spw) 167
7.20.4.12 chSemFastWaitI(semaphore_t *sp) 168
7.20.4.13 chSemFastSignalI(semaphore_t *sp) 169
7.20.4.14 chSemGetCounterI(const semaphore_t *sp) 169

7.21 Mutexes 171

7.21.1 Detailed Description 171
7.21.2 Macro Definition Documentation 172
7.21.2.1 _MUTEX_DATA 172
7.21.2.2 MUTEX_DECL 173
7.21.3 Typedef Documentation 173
7.21.3.1 mutex_t 173
7.21.4 Function Documentation 173
7.21.4.1 chMtxObjectInit(mutex_t *mp) 173
7.21.4.2 chMtxLock(mutex_t *mp) 173
7.21.4.3 chMtxLockS(mutex_t *mp) 174
7.21.4.4 chMtxTryLock(mutex_t *mp) 175
7.21.4.5 chMtxTryLockS(mutex_t *mp) 176
7.21.4.6 chMtxUnlock(mutex_t *mp) 177
7.21.4.7 chMtxUnlockS(mutex_t *mp) 178
7.21.4.8 chMtxUnlockAllS(void) 179
7.21.4.9 chMtxUnlockAll(void) 179
7.21.4.10 chMtxQueueNotEmptyS(mutex_t *mp) 180
7.21.4.11 chMtxGetOwnerI(mutex_t *mp) 181
7.21.4.12 chMtxGetNextMutexX(void) 182

7.22 Condition Variables 183

7.22.1 Detailed Description 183
7.22.2 Macro Definition Documentation 184
7.22.2.1 _CONDVAR_DATA 184
7.22.2.2 CONDVAR_DECL 184
7.22.3 Typedef Documentation 184
7.22.3.1 condition_variable_t 184
7.22.4 Function Documentation 184
7.22.4.1 chCondObjectInit(condition_variable_t *cp) 184
7.22.4.2 chCondSignal(condition_variable_t *cp) 185
7.22.4.3 chCondSignalI(condition_variable_t *cp) 186
7.22.4.4 chCondBroadcast(condition_variable_t *cp) 187
7.23.4.23 chEvtBroadcast(event_source_t *.esp) ............................................. 212
7.23.4.24 chEvtBroadcastI(event_source_t *esp) .............................................. 213
7.23.4.25 chEvtAddEventsI(eventmask_t events) ................................................ 213
7.23.4.26 chEvtGetEventsX(void) .................................................................... 214

7.24 Synchronous Messages ................................................................. 215
7.24.1 Detailed Description ............................................................... 215
7.24.2 Function Documentation ......................................................... 215
7.24.2.1 chMsgSend(thread_t *tp, msg_t msg) ................................................. 215
7.24.2.2 chMsgWait(void) ....................................................................... 216
7.24.2.3 chMsgRelease(thread_t *tp, msg_t msg) ........................................... 217
7.24.2.4 chMsgIsPendingI(thread_t *tp) ......................................................... 218
7.24.2.5 chMsgGet(thread_t *tp) .................................................................. 219
7.24.2.6 chMsgReleaseS(thread_t *tp, msg_t msg) ........................................ 219

7.25 Dynamic Threads ........................................................................ 220
7.25.1 Detailed Description ............................................................... 220
7.25.2 Function Documentation ......................................................... 220
7.25.2.1 chThdCreateFromHeap(memory_heap_t *heapp, size_t size, const char *name, tprio_t prio, tfunc_t pf, void *arg) .......................................................... 220
7.25.2.2 chThdCreateFromMemoryPool(memory_pool_t *mp, const char *name, tprio_t prio, tfunc_t pf, void *arg) ................................................................. 221

7.26 Registry .................................................................................... 223
7.26.1 Detailed Description ............................................................... 223
7.26.2 Macro Definition Documentation ............................................. 224
7.26.2.1 REG_REMOVE ........................................................................... 224
7.26.2.2 REG_INSERT ............................................................................ 224
7.26.3 Function Documentation .......................................................... 224
7.26.3.1 chRegFirstThread(void) ................................................................. 224
7.26.3.2 chRegNextThread(thread_t *tp) ....................................................... 225
7.26.3.3 chRegFindThreadByName(const char *name) .................................. 226
7.26.3.4 chRegFindThreadByPointer(thread_t *tp) ........................................ 227
7.26.3.5 chRegFindThreadByWorkingArea(stkalign_t *wa) ................................. 228
7.26.3.6 chRegSetName(thread_t *tp, const char *name) .............................. 228
7.26.3.7 chRegGetThreadNameX(thread_t *tp) .................................................. 229
7.26.3.8 chRegSetThreadNameX(thread_t *tp, const char *name) ...................... 229

7.27 Debug ....................................................................................... 230
7.27.1 Detailed Description ............................................................... 230

7.28 Checks and Assertions ................................................................. 231
7.28.1 Detailed Description ............................................................... 231
7.28.2 Macro Definition Documentation ............................................. 232
7.28.2.1 CH_DBG_STACK_FILL_VALUE ....................................................... 232
7.30.2.6 _stats_start_measure_crit_isr(void) ................................................. 249
7.30.2.7 _stats_stop_measure_crit_isr(void) .................................................. 249
7.31 OS Library .......................................................... 250
7.31.1 Detailed Description ......................................................... 250
7.32 Version Numbers and Identification ..................................................... 251
7.32.1 Detailed Description .......................................................... 251
7.32.2 Macro Definition Documentation ...................................................... 251
7.32.2.1 _CHIBIOS_OSLIB_ .......................................................... 251
7.32.2.2 CH_OSLIB_STABLE .......................................................... 251
7.32.2.3 CH_OSLIB_VERSION ......................................................... 251
7.32.2.4 CH_OSLIB_MAJOR ......................................................... 251
7.32.2.5 CH_OSLIB_MINOR ......................................................... 251
7.32.2.6 CH_OSLIB_PATCH ......................................................... 251
7.33 Synchronization .......................................................... 252
7.33.1 Detailed Description .......................................................... 252
7.34 Binary Semaphores .......................................................... 253
7.34.1 Detailed Description .......................................................... 253
7.34.2 Macro Definition Documentation ...................................................... 253
7.34.2.1 _BSEMAPHORE_DATA .................................................. 253
7.34.2.2 BSEMAPHORE_DECL ....................................................... 254
7.34.3 Typedef Documentation .......................................................... 254
7.34.3.1 binary_semaphore_t ......................................................... 254
7.34.4 Function Documentation .......................................................... 254
7.34.4.1 chBSemObjectInit(binary_semaphore_t *bsp, bool taken) .................. 254
7.34.4.2 chBSemWait(binary_semaphore_t *bsp) ...................................... 255
7.34.4.3 chBSemWaitS(binary_semaphore_t *bsp) ..................................... 255
7.34.4.4 chBSemWaitTimeoutS(binary_semaphore_t *bsp, sysinterval_t timeout) .... 256
7.34.4.5 chBSemWaitTimeout(binary_semaphore_t *bsp, sysinterval_t timeout) .... 257
7.34.4.6 chBSemResetI(binary_semaphore_t *bsp, bool taken) ..................... 258
7.34.4.7 chBSemReset(binary_semaphore_t *bsp, bool taken) ....................... 259
7.34.4.8 chBSemSignalI(binary_semaphore_t *bsp) .................................... 259
7.34.4.9 chBSemSignal(binary_semaphore_t *bsp) ..................................... 260
7.34.4.10 chBSemGetStateI(const binary_semaphore_t *bsp) ....................... 261
7.35 Mailboxes ............................................................ 263
7.35.1 Detailed Description .......................................................... 263
7.35.2 Macro Definition Documentation ...................................................... 264
7.35.2.1 _MAILBOX_DATA .......................................................... 264
7.35.2.2 MAILBOX_DECL .......................................................... 265
7.35.3 Function Documentation .......................................................... 265
7.35.3.1 chMBOBJECTInit(mailbox_t *mbp, msg_t *buf, size_t n) .................... 265
7.35.3.2 chMBReset(mailbox_t *mbp) ............................................. 265
7.35.3.3 chMBResetI(mailbox_t *mbp) ........................................ 266
7.35.3.4 chMBPostTimeout(mailbox_t *mbp, msg_t msg, sysinterval_t timeout) .............................. 267
7.35.3.5 chMBPostTimeoutS(mailbox_t *mbp, msg_t msg, sysinterval_t timeout) ......................... 268
7.35.3.6 chMBPostI(mailbox_t *mbp, msg_t msg) ..................................... 269
7.35.3.7 chMBPostAheadTimeout(mailbox_t *mbp, msg_t msg, sysinterval_t timeout) .................. 270
7.35.3.8 chMBPostAheadTimeoutS(mailbox_t *mbp, msg_t msg, sysinterval_t timeout) ................ 271
7.35.3.9 chMBPostAheadI(mailbox_t *mbp, msg_t msg) .................................. 271
7.35.3.10 chMBFetchTimeout(mailbox_t *mbp, msg_t *msgp, sysinterval_t timeout) ................. 272
7.35.3.11 chMBFetchTimeoutS(mailbox_t *mbp, msg_t *msgp, sysinterval_t timeout) ............... 272
7.35.3.12 chMBFetchI(mailbox_t *mbp, msg_t *msgp) ..................................... 274
7.35.3.13 chMBGetSizeI(const mailbox_t *mbp) ........................................ 275
7.35.3.14 chMBGetUsedCountI(const mailbox_t *mbp) .................................... 275
7.35.3.15 chMBGetFreeCountI(const mailbox_t *mbp) ....................................... 276
7.35.3.16 chMBPeekI(const mailbox_t *mbp) ............................................. 276
7.35.3.17 chMBResumeX(mailbox_t *mbp) ................................................ 277

7.36 Pipes ................................................................. 278
7.36.1 Detailed Description .................................................... 278
7.36.2 Macro Definition Documentation ...................................... 278
7.36.2.1 _PIPE_DATA ................................................................. 278
7.36.2.2 PIPE_DECL ................................................................. 279
7.36.3 Function Documentation ................................................. 279
7.36.3.1 pipe_write(pipe_t *pp, const uint8_t *bp, size_t n) ................ 279
7.36.3.2 pipe_read(pipe_t *pp, uint8_t *bp, size_t n) ............................. 280
7.36.3.3 chPipeObjectInit(pipe_t *pp, uint8_t *buf, size_t n) ................. 281
7.36.3.4 chPipeReset(pipe_t *pp) .................................................. 281
7.36.3.5 chPipeWriteTimeout(pipe_t *pp, const uint8_t *bp, size_t n, sysinterval_t timeout) ... 282
7.36.3.6 chPipeReadTimeout(pipe_t *pp, uint8_t *bp, size_t n, sysinterval_t timeout) ........... 283
7.36.3.7 chPipeGetSize(const pipe_t *pp) ............................................ 284
7.36.3.8 chPipeGetUsedCount(const pipe_t *pp) ................................... 284
7.36.3.9 chPipeGetFreeCount(const pipe_t *pp) ..................................... 285
7.36.3.10 chPipeResume(pipe_t *pp) .................................................. 285

7.37 Memory Management .................................................... 286
7.37.1 Detailed Description .................................................... 286
7.38 Core Memory Manager ................................................... 287
7.38.1 Detailed Description .................................................... 287
7.38.2 Macro Definition Documentation ...................................... 288
7.38.2.1 CH_CFG_MEMCORE_SIZE .................................................. 288
7.38.3 Typedef Documentation .................................................. 288
7.38.3.1 memgetfunc_t ............................................................... 288
7.38.3.2 memgetfunc2_t .......................................................... 288
7.38.4 Function Documentation ............................................... 288
  7.38.4.1 _core_init(void) ............................................................ 288
  7.38.4.2 chCoreAllocAlignedWithOffset(size_t size, unsigned align, size_t offset) ........................................ 289
  7.38.4.3 chCoreAllocAlignedWithOffset(size_t size, unsigned align, size_t offset) ........................................ 289
  7.38.4.4 chCoreGetStatusX(void) ................................................. 290
  7.38.4.5 chCoreAllocAligned(size_t size, unsigned align) .......... 290
  7.38.4.6 chCoreAllocAligned(size_t size, unsigned align) .......... 291
  7.38.4.7 chCoreAllocI(size_t size) ............................................. 292
  7.38.4.8 chCoreAlloc(size_t size) ............................................. 293
7.38.5 Variable Documentation .............................................. 293
  7.38.5.1 ch_memcore .............................................................. 293
7.39 Memory Heaps .............................................................. 294
  7.39.1 Detailed Description .................................................... 294
  7.39.2 Macro Definition Documentation ..................................... 295
    7.39.2.1 CH_HEAP_ALIGNMENT ................................................. 295
    7.39.2.2 CH_HEAP_AREA .................................................... 295
  7.39.3 Typedef Documentation .............................................. 295
    7.39.3.1 memory_heap_t .................................................... 295
    7.39.3.2 heap_header_t .................................................... 295
  7.39.4 Function Documentation ............................................ 295
    7.39.4.1 _heap_init(void) .................................................. 295
    7.39.4.2 chHeapObjectInit(memory_heap_t *heapp, void *buf, size_t size) .................................................. 296
    7.39.4.3 chHeapAllocAligned(memory_heap_t *heapp, size_t size, unsigned align) ........................................ 296
    7.39.4.4 chHeapFree(void *p) .............................................. 297
    7.39.4.5 chHeapStatus(memory_heap_t *heapp, size_t *totalp, size_t *largestp) ........................................ 297
    7.39.4.6 chHeapAlloc(memory_heap_t *heapp, size_t size) .......... 298
    7.39.4.7 chHeapGetSize(const void *p) ................................... 298
  7.39.5 Variable Documentation ............................................ 299
    7.39.5.1 default_heap ...................................................... 299
7.40 Memory Pools .............................................................. 300
  7.40.1 Detailed Description .................................................. 300
  7.40.2 Macro Definition Documentation ..................................... 301
    7.40.2.1 _MEMORYPOOL_DATA ............................................. 301
    7.40.2.2 MEMORYPOOL_DECL ................................................ 302
    7.40.2.3 _GUARDEDMEMORYPOOL_DATA .................................... 302
    7.40.2.4 GUARDEDMEMORYPOOL_DECL ..................................... 302
  7.40.3 Function Documentation ............................................ 302
    7.40.3.1 chPoolObjectInitAligned(memory_pool_t *mp, size_t size, unsigned align, memgetfunc_t provider) ................... 302
7.40.3.2 chPoolLoadArray(memory_pool_t *mp, void *p, size_t n) .......................... 303
7.40.3.3 chPoolAlloc(memory_pool_t *mp) ............................................................ 303
7.40.3.4 chPoolAlloc(memory_pool_t *mp) ............................................................ 304
7.40.3.5 chPoolFree(memory_pool_t *mp, void *objp) .......................................... 305
7.40.3.6 chPoolFree(memory_pool_t *mp, void *objp) .......................................... 306
7.40.3.7 chGuardedPoolObjectInitAligned(guarded_memory_pool_t *gmp, size_t size, unsigned align) .......................................................... 307
7.40.3.8 chGuardedPoolLoadArray(guarded_memory_pool_t *gmp, void *p, size_t n) 307
7.40.3.9 chGuardedPoolAllocTimeoutS(guarded_memory_pool_t *gmp, sysinterval_t timeout) .......................................................... 307
7.40.3.10 chGuardedPoolAllocTimeout(guarded_memory_pool_t *gmp, sysinterval_t timeout) .......................................................... 308
7.40.3.11 chGuardedPoolFree(guarded_memory_pool_t *gmp, void *objp) ................. 308
7.40.3.12 chPoolObjectInit(memory_pool_t *mp, size_t size, memgetfunc_t provider) 310
7.40.3.13 chPoolAdd(memory_pool_t *mp, void *objp) ............................................. 312
7.40.3.14 chPoolAddI(memory_pool_t *mp, void *objp) ........................................... 312
7.40.3.15 chGuardedPoolObjectInit(guarded_memory_pool_t *gmp, size_t size) .......... 312
7.40.3.16 chGuardedPoolGetCounterI(guarded_memory_pool_t *gmp) ...................... 313
7.40.3.17 chGuardedPoolAllocI(guarded_memory_pool_t *gmp) .............................. 313
7.40.3.18 chGuardedPoolFreeI(guarded_memory_pool_t *gmp, void *objp) ............... 314
7.40.3.19 chGuardedPoolFreeS(guarded_memory_pool_t *gmp, void *objp) ............... 315
7.40.3.20 chGuardedPoolAdd(guarded_memory_pool_t *gmp, void *objp) ................. 316
7.40.3.21 chGuardedPoolAddI(guarded_memory_pool_t *gmp, void *objp) ................. 316
7.40.3.22 chGuardedPoolAddS(guarded_memory_pool_t *gmp, void *objp) ................. 317

7.41 Complex Services ................................................................. 319
7.41.1 Detailed Description ............................................................ 319

7.42 Objects FIFOs ................................................................. 320
7.42.1 Detailed Description ............................................................ 320
7.42.2 Typedef Documentation ........................................................ 321
7.42.2.1 objects_fifo_t ............................................................................ 321
7.42.3 Function Documentation .......................................................... 321
7.42.3.1 chFifoObjectInitAligned(objects_fifo_t *ofp, size_t objsize, size_t objn, unsigned objalign, void *objbuf, msg_t *msgbuf) .................. 321
7.42.3.2 chFifoObjectInit(objects_fifo_t *ofp, size_t objsize, size_t objn, void *objbuf, msg_t *msgbuf) .......................................................... 321
7.42.3.3 chFifoTakeObjectI(objects_fifo_t *ofp) ............................................. 322
7.42.3.4 chFifoTakeObjectTimeoutI(objects_fifo_t *ofp, sysinterval_t timeout) ........ 323
7.42.3.5 chFifoTakeObjectTimeout(objects_fifo_t *ofp, sysinterval_t timeout) .......... 323
7.42.3.6 chFifoReturnObjectI(objects_fifo_t *ofp, void *objp) ......................... 324
7.42.3.7 chFifoReturnObjectI(objects_fifo_t *ofp, void *objp) ......................... 325
7.42.3.8 chFifoReturnObjectI(objects_fifo_t *ofp, void *objp) ......................... 325
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.42.3.9</td>
<td>chFifoSendObjectI(objects_fifo_t *ofp, void *objp)</td>
<td>326</td>
</tr>
<tr>
<td>7.42.3.10</td>
<td>chFifoSendObjectS(objects_fifo_t *ofp, void *objp)</td>
<td>326</td>
</tr>
<tr>
<td>7.42.3.11</td>
<td>chFifoSendObject(objects_fifo_t *ofp, void *objp)</td>
<td>327</td>
</tr>
<tr>
<td>7.42.3.12</td>
<td>chFifoSendObjectAheadI(objects_fifo_t *ofp, void *objp)</td>
<td>328</td>
</tr>
<tr>
<td>7.42.3.13</td>
<td>chFifoSendObjectAheadS(objects_fifo_t *ofp, void *objp)</td>
<td>328</td>
</tr>
<tr>
<td>7.42.3.14</td>
<td>chFifoSendObjectAhead(objects_fifo_t *ofp, void *objp)</td>
<td>329</td>
</tr>
<tr>
<td>7.42.3.15</td>
<td>chFifoReceiveObjectI(objects_fifo_t *ofp, void **objpp)</td>
<td>330</td>
</tr>
<tr>
<td>7.42.3.16</td>
<td>chFifoReceiveObjectTimeoutS(objects_fifo_t *ofp, void **objpp, sysinterval_t timeout)</td>
<td>330</td>
</tr>
<tr>
<td>7.42.3.17</td>
<td>chFifoReceiveObjectTimeout(objects_fifo_t *ofp, void **objpp, sysinterval_t timeout)</td>
<td>331</td>
</tr>
<tr>
<td>7.43</td>
<td>Dynamic Objects Factory</td>
<td>333</td>
</tr>
<tr>
<td>7.43.1</td>
<td>Detailed Description</td>
<td>333</td>
</tr>
<tr>
<td>7.43.2</td>
<td>Macro Definition Documentation</td>
<td>336</td>
</tr>
<tr>
<td>7.43.2.1</td>
<td>CH_CFG_FACTORY_MAX_NAMES_LENGTH</td>
<td>336</td>
</tr>
<tr>
<td>7.43.2.2</td>
<td>CH_CFG_FACTORY_OBJECTS_REGISTRY</td>
<td>336</td>
</tr>
<tr>
<td>7.43.2.3</td>
<td>CH_CFG_FACTORY_GENERIC_BUFFERS</td>
<td>336</td>
</tr>
<tr>
<td>7.43.2.4</td>
<td>CH_CFG_FACTORY_SEMAPHORES</td>
<td>336</td>
</tr>
<tr>
<td>7.43.2.5</td>
<td>CH_CFG_FACTORY_SEMAPHORES</td>
<td>336</td>
</tr>
<tr>
<td>7.43.2.6</td>
<td>CH_CFG_FACTORY_MAILBOXES</td>
<td>336</td>
</tr>
<tr>
<td>7.43.2.7</td>
<td>CH_CFG_FACTORY_MAILBOXES</td>
<td>336</td>
</tr>
<tr>
<td>7.43.2.8</td>
<td>CH_CFG_FACTORY_OBJ_FIFOS</td>
<td>336</td>
</tr>
<tr>
<td>7.43.2.9</td>
<td>CH_CFG_FACTORY_OBJ_FIFOS</td>
<td>336</td>
</tr>
<tr>
<td>7.43.2.10</td>
<td>CH_CFG_FACTORY_OBJ_FIFOS</td>
<td>336</td>
</tr>
<tr>
<td>7.43.2.11</td>
<td>CH_CFG_FACTORY_PIPES</td>
<td>337</td>
</tr>
<tr>
<td>7.43.2.12</td>
<td>CH_CFG_FACTORY_PIPES</td>
<td>337</td>
</tr>
<tr>
<td>7.43.3</td>
<td>Typedef Documentation</td>
<td>337</td>
</tr>
<tr>
<td>7.43.3.1</td>
<td>dyn_element_t</td>
<td>337</td>
</tr>
<tr>
<td>7.43.3.2</td>
<td>dyn_list_t</td>
<td>337</td>
</tr>
<tr>
<td>7.43.3.3</td>
<td>registered_object_t</td>
<td>337</td>
</tr>
<tr>
<td>7.43.3.4</td>
<td>dyn_buffer_t</td>
<td>337</td>
</tr>
<tr>
<td>7.43.3.5</td>
<td>dyn_semaphore_t</td>
<td>337</td>
</tr>
<tr>
<td>7.43.3.6</td>
<td>dyn_mailbox_t</td>
<td>337</td>
</tr>
<tr>
<td>7.43.3.7</td>
<td>dyn_objects_fifo_t</td>
<td>337</td>
</tr>
<tr>
<td>7.43.3.8</td>
<td>dyn_pipe_t</td>
<td>337</td>
</tr>
<tr>
<td>7.43.3.9</td>
<td>objects_factory_t</td>
<td>337</td>
</tr>
<tr>
<td>7.43.4</td>
<td>Function Documentation</td>
<td>338</td>
</tr>
<tr>
<td>7.43.4.1</td>
<td>_factory_init(void)</td>
<td>338</td>
</tr>
<tr>
<td>7.43.4.2</td>
<td>chFactoryRegisterObject(const char *name, void *objp)</td>
<td>338</td>
</tr>
<tr>
<td>7.43.4.3</td>
<td>chFactoryFindObject(const char *name)</td>
<td>339</td>
</tr>
</tbody>
</table>
7.43.4 chFactoryFindObjectByPointer(void *objp) ........................................... 339
7.43.5 chFactoryReleaseObject(registered_object_t *rop) .................................. 340
7.43.6 chFactoryCreateBuffer(const char *name, size_t size) ............................. 340
7.43.7 chFactoryFindBuffer(const char *name) .................................................. 340
7.43.8 chFactoryReleaseBuffer(dyn_buffer_t *dbp) ............................................ 341
7.43.9 chFactoryCreateSemaphore(const char *name, cnt_t n) ............................. 341
7.43.10 chFactoryFindSemaphore(const char *name) ........................................... 342
7.43.11 chFactoryReleaseSemaphore(dyn_semaphore_t *dsp) ................................ 342
7.43.12 chFactoryCreateMailbox(const char *name, size_t n) .............................. 343
7.43.13 chFactoryFindMailbox(const char *name) .............................................. 343
7.43.14 chFactoryReleaseMailbox(dyn_mailbox_t *dmp) ....................................... 344
7.43.15 chFactoryCreateObjectsFIFO(const char *name, size_t objsize, size_t objn, unsigned objalign) ......................................................... 344
7.43.16 chFactoryFindObjectesFIFO(const char *name) ....................................... 345
7.43.17 chFactoryReleaseObjectsFIFO(dyn_objects_fifo_t *dofp) ......................... 346
7.43.18 chFactoryCreatePipe(const char *name, size_t size) ................................ 346
7.43.19 chFactoryFindPipe(const char *name) ................................................... 347
7.43.20 chFactoryReleasePipe(dyn_pipe_t *dpp) ................................................ 347
7.43.21 chFactoryDuplicateReference(dyn_element_t *dep) ................................ 347
7.43.22 chFactoryGetObject(registered_object_t *rop) ...................................... 348
7.43.23 chFactoryGetBufferSize(dyn_buffer_t *dbp) ......................................... 348
7.43.24 chFactoryGetBuffer(dyn_buffer_t *dbp) ............................................... 348
7.43.25 chFactoryGetSemaphore(dyn_semaphore_t *dsp) ..................................... 349
7.43.26 chFactoryGetMailbox(dyn_mailbox_t *dmp) ............................................ 349
7.43.27 chFactoryGetObjectsFIFO(dyn_objects_fifo_t *dofp) ............................... 349
7.43.28 chFactoryGetPipe(dyn_pipe_t *dpp) ...................................................... 350
7.43.5 Variable Documentation ................................................................. 350
7.43.5.1 ch_factory ................................................................. 350

8 Data Structure Documentation ..................................................... 351
8.1 ch_binary_semaphore Struct Reference .................................................. 351
8.1.1 Detailed Description ........................................................................ 352
8.2 ch_dyn_element Struct Reference .......................................................... 352
8.2.1 Detailed Description ........................................................................ 353
8.2.2 Field Documentation ....................................................................... 353
8.2.2.1 next ......................................................................................... 353
8.2.2.2 refs ......................................................................................... 353
8.3 ch_dyn_list Struct Reference ................................................................. 353
8.3.1 Detailed Description ....................................................................... 354
8.4 ch_dyn_mailbox Struct Reference .......................................................... 354
8.4.1 Detailed Description .......................................................... 355
8.4.2 Field Documentation ....................................................... 355
  8.4.2.1 element ................................................................. 355
  8.4.2.2 mbx ................................................................. 356
  8.4.2.3 msgbuf ............................................................... 356
8.5 ch_dyn_object Struct Reference ........................................ 356
  8.5.1 Detailed Description .................................................. 356
  8.5.2 Field Documentation .................................................. 357
    8.5.2.1 element .......................................................... 357
    8.5.2.2 buffer .......................................................... 357
8.6 ch_dyn_objects_fifo Struct Reference .................................. 357
  8.6.1 Detailed Description .................................................. 358
  8.6.2 Field Documentation .................................................. 358
    8.6.2.1 element .......................................................... 358
    8.6.2.2 fifo .............................................................. 358
    8.6.2.3 msgbuf .......................................................... 358
8.7 ch_dyn_pipe Struct Reference ............................................ 359
  8.7.1 Detailed Description .................................................. 361
  8.7.2 Field Documentation .................................................. 361
    8.7.2.1 element .......................................................... 361
    8.7.2.2 pipe ............................................................. 361
    8.7.2.3 buffer .......................................................... 361
8.8 ch_dyn_semaphore Struct Reference ...................................... 361
  8.8.1 Detailed Description .................................................. 362
  8.8.2 Field Documentation .................................................. 362
    8.8.2.1 element .......................................................... 362
    8.8.2.2 sem .............................................................. 362
8.9 ch_mutex Struct Reference .................................................. 363
  8.9.1 Detailed Description .................................................. 364
  8.9.2 Field Documentation .................................................. 364
    8.9.2.1 queue ........................................................... 364
    8.9.2.2 owner ........................................................... 364
    8.9.2.3 next ............................................................. 364
    8.9.2.4 cnt .............................................................. 364
8.10 ch_objects_factory Struct Reference .................................... 364
  8.10.1 Detailed Description .................................................. 366
  8.10.2 Field Documentation .................................................. 366
    8.10.2.1 mtx ............................................................. 366
    8.10.2.2 obj_list ...................................................... 366
    8.10.2.3 obj_pool ....................................................... 366
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>8.16.2.1</td>
<td>queue</td>
<td>377</td>
</tr>
<tr>
<td>8.16.2.2</td>
<td>prio</td>
<td>377</td>
</tr>
<tr>
<td>8.16.2.3</td>
<td>ctx</td>
<td>377</td>
</tr>
<tr>
<td>8.16.2.4</td>
<td>newer</td>
<td>377</td>
</tr>
<tr>
<td>8.16.2.5</td>
<td>older</td>
<td>377</td>
</tr>
<tr>
<td>8.16.2.6</td>
<td>name</td>
<td>377</td>
</tr>
<tr>
<td>8.16.2.7</td>
<td>wabase</td>
<td>377</td>
</tr>
<tr>
<td>8.16.2.8</td>
<td>state</td>
<td>377</td>
</tr>
<tr>
<td>8.16.2.9</td>
<td>flags</td>
<td>377</td>
</tr>
<tr>
<td>8.16.2.10</td>
<td>refs</td>
<td>378</td>
</tr>
<tr>
<td>8.16.2.11</td>
<td>ticks</td>
<td>378</td>
</tr>
<tr>
<td>8.16.2.12</td>
<td>time</td>
<td>378</td>
</tr>
<tr>
<td>8.16.2.13</td>
<td>rdymsg</td>
<td>378</td>
</tr>
<tr>
<td>8.16.2.14</td>
<td>exitcode</td>
<td>378</td>
</tr>
<tr>
<td>8.16.2.15</td>
<td>wtobjp</td>
<td>378</td>
</tr>
<tr>
<td>8.16.2.16</td>
<td>wtrp</td>
<td>378</td>
</tr>
<tr>
<td>8.16.2.17</td>
<td>sentmsg</td>
<td>379</td>
</tr>
<tr>
<td>8.16.2.18</td>
<td>wttmp</td>
<td>379</td>
</tr>
<tr>
<td>8.16.2.19</td>
<td>wtmxp</td>
<td>379</td>
</tr>
<tr>
<td>8.16.2.20</td>
<td>ewmask</td>
<td>379</td>
</tr>
<tr>
<td>8.16.2.21</td>
<td>u</td>
<td>379</td>
</tr>
<tr>
<td>8.16.2.22</td>
<td>waiting</td>
<td>379</td>
</tr>
<tr>
<td>8.16.2.23</td>
<td>msgqueue</td>
<td>379</td>
</tr>
<tr>
<td>8.16.2.24</td>
<td>epending</td>
<td>379</td>
</tr>
<tr>
<td>8.16.2.25</td>
<td>mtxlist</td>
<td>380</td>
</tr>
<tr>
<td>8.16.2.26</td>
<td>realprio</td>
<td>380</td>
</tr>
<tr>
<td>8.16.2.27</td>
<td>mpool</td>
<td>380</td>
</tr>
<tr>
<td>8.16.2.28</td>
<td>stats</td>
<td>380</td>
</tr>
<tr>
<td>8.17</td>
<td>ch_threads_list Struct Reference</td>
<td>380</td>
</tr>
<tr>
<td>8.17.1</td>
<td>Detailed Description</td>
<td>382</td>
</tr>
<tr>
<td>8.17.2</td>
<td>Field Documentation</td>
<td>382</td>
</tr>
<tr>
<td>8.17.2.1</td>
<td>next</td>
<td>382</td>
</tr>
<tr>
<td>8.18</td>
<td>ch_threads_queue Struct Reference</td>
<td>382</td>
</tr>
<tr>
<td>8.18.1</td>
<td>Detailed Description</td>
<td>383</td>
</tr>
<tr>
<td>8.18.2</td>
<td>Field Documentation</td>
<td>383</td>
</tr>
<tr>
<td>8.18.2.1</td>
<td>next</td>
<td>383</td>
</tr>
<tr>
<td>8.18.2.2</td>
<td>prev</td>
<td>383</td>
</tr>
<tr>
<td>8.19</td>
<td>ch_trace_buffer_t Struct Reference</td>
<td>383</td>
</tr>
<tr>
<td>8.19.1</td>
<td>Detailed Description</td>
<td>385</td>
</tr>
<tr>
<td>8.19.2</td>
<td>Field Documentation</td>
<td>385</td>
</tr>
</tbody>
</table>
8.19.2.1 suspended .................................................. 385
8.19.2.2 size .................................................... 385
8.19.2.3 ptr ..................................................... 385
8.19.2.4 buffer ................................................ 385

8.20 ch_trace_event_t Struct Reference .................................................. 385
  8.20.1 Detailed Description ........................................ 387
  8.20.2 Field Documentation ........................................ 387
    8.20.2.1 type .................................................. 387
    8.20.2.2 state ................................................ 387
    8.20.2.3 rtstamp .............................................. 388
    8.20.2.4 time ................................................ 388
    8.20.2.5 ntp .................................................. 388
    8.20.2.6 wtobjp .............................................. 388
    8.20.2.7 sw ................................................... 388
    8.20.2.8 name ................................................. 388
    8.20.2.9 isr .................................................. 388
    8.20.2.10 reason .............................................. 388
    8.20.2.11 halt ............................................... 388
    8.20.2.12 up1 ............................................... 388
    8.20.2.13 up2 ............................................... 388
    8.20.2.14 user ............................................... 389

8.21 ch_virtual_timer Struct Reference .............................................. 389
  8.21.1 Detailed Description ........................................ 390
  8.21.2 Field Documentation ........................................ 390
    8.21.2.1 next .................................................. 390
    8.21.2.2 prev ................................................ 390
    8.21.2.3 delta ............................................... 391
    8.21.2.4 func ............................................... 391
    8.21.2.5 par .................................................. 391

8.22 ch_virtual_timers_list Struct Reference ...................................... 391
  8.22.1 Detailed Description ...................................... 392
  8.22.2 Field Documentation ...................................... 392
    8.22.2.1 next .................................................. 392
    8.22.2.2 prev ................................................ 392
    8.22.2.3 delta ............................................... 393
    8.22.2.4 systime ............................................. 393
    8.22.2.5 lasttime .......................................... 393

8.23 chdebug_t Struct Reference ................................................. 393
  8.23.1 Detailed Description ...................................... 394
  8.23.2 Field Documentation ...................................... 394
8.28 Detailed Description .................................................. 401
8.28.1 next ................................................................. 401
8.28.2 pages ............................................................... 401
8.28.3 heap ................................................................. 401
8.28.4 size ................................................................. 402
8.29 kernel_stats_t Struct Reference ................................. 402
8.29.1 n_irq ............................................................ 403
8.29.2 n_ctxswc ....................................................... 403
8.29.3 m_crit_thd ..................................................... 403
8.29.4 m_crit_isr ..................................................... 403
8.30 mailbox_t Struct Reference ................................. 404
8.30.1 buffer .......................................................... 404
8.30.2 top .............................................................. 404
8.30.3 wrptr .......................................................... 404
8.30.4 rdptr ........................................................... 405
8.30.5 cnt ............................................................. 405
8.30.6 reset ........................................................... 405
8.30.7 qw ............................................................. 405
8.30.8 qr .............................................................. 405
8.31 memcore_t Struct Reference ................................. 406
8.31.1 nextmem ......................................................... 406
8.31.2 endmem .......................................................... 406
8.32 memory_heap Struct Reference ................................. 407
8.32.1 provider .......................................................... 407
8.32.2 header ............................................................ 407
8.32.3 mtx ............................................................... 407
8.33 memory_pool_t Struct Reference ................................. 408
8.33.1 next .............................................................. 408
8.33.2 object_size .......................................................... 408
8.33.2.3 align .................................................. 408
8.33.2.4 provider ............................................. 408
8.34 pipe_t Struct Reference .................................. 408
  8.34.1 Detailed Description ................................ 410
  8.34.2 Field Documentation ................................ 410
    8.34.2.1 buffer ............................................ 410
    8.34.2.2 top ................................................. 410
    8.34.2.3 wrptr .............................................. 410
    8.34.2.4 rdptr .............................................. 410
    8.34.2.5 cnt ................................................ 410
    8.34.2.6 reset .............................................. 410
    8.34.2.7 wrt ................................................ 411
    8.34.2.8 rtr ................................................ 411
    8.34.2.9 cmtx ............................................... 411
    8.34.2.10 wmtx .............................................. 411
    8.34.2.11 rmtx .............................................. 411
8.35 pool_header Struct Reference .......................... 411
  8.35.1 Detailed Description ................................ 411
  8.35.2 Field Documentation ................................ 412
    8.35.2.1 next ............................................... 412
8.36 thread_descriptor_t Struct Reference ................ 412
  8.36.1 Detailed Description ................................ 412
  8.36.2 Field Documentation ................................ 413
    8.36.2.1 name .............................................. 413
    8.36.2.2 wbase ............................................ 413
    8.36.2.3 wend .............................................. 413
    8.36.2.4 prio ............................................... 413
    8.36.2.5 funcp ............................................. 413
    8.36.2.6 arg ................................................ 413
8.37 time_measurement_t Struct Reference ................ 413
  8.37.1 Detailed Description ................................ 414
  8.37.2 Field Documentation ................................ 414
    8.37.2.1 best ............................................... 414
    8.37.2.2 worst ............................................... 414
    8.37.2.3 last ............................................... 414
    8.37.2.4 n .................................................. 414
    8.37.2.5 cumulative ....................................... 414
8.38 tm_calibration_t Struct Reference .................... 414
  8.38.1 Detailed Description ................................ 415
  8.38.2 Field Documentation ................................ 415
9 File Documentation

9.1 ch.h File Reference ........................................ 417
  9.1.1 Detailed Description ................................ 418
9.2 chalign.h File Reference ................................. 418
  9.2.1 Detailed Description ................................ 418
9.3 chbsem.h File Reference ................................. 418
  9.3.1 Detailed Description ................................ 419
9.4 chchecks.h File Reference ................................ 420
  9.4.1 Detailed Description ................................ 420
9.5 chcond.c File Reference .................................. 420
  9.5.1 Detailed Description ................................ 421
9.6 chcond.h File Reference .................................. 421
  9.6.1 Detailed Description ................................ 422
9.7 chconf.h File Reference .................................. 422
  9.7.1 Detailed Description ................................ 424
9.8 chcustomer.h File Reference ......................... 424
  9.8.1 Detailed Description ................................ 425
9.9 chdebug.c File Reference .............................. 425
  9.9.1 Detailed Description ................................ 426
9.10 chdebug.h File Reference ............................. 426
   9.10.1 Detailed Description ............................... 426
9.11 chdynamic.c File Reference .................... 426
   9.11.1 Detailed Description ............................... 426
9.12 chdynamic.h File Reference ..................... 426
   9.12.1 Detailed Description ............................... 427
9.13 chevents.c File Reference ......................... 427
   9.13.1 Detailed Description ............................... 428
9.14 chevents.h File Reference ......................... 428
   9.14.1 Detailed Description ............................... 428
9.15 chfactory.c File Reference ..................... 430
   9.15.1 Detailed Description ............................... 431
9.16 chfactory.h File Reference ..................... 431
   9.16.1 Detailed Description ............................... 431
9.17 chlib.h File Reference ......................... 434
   9.17.1 Detailed Description ............................... 434
9.18 chlicense.h File Reference .................. 435
   9.18.1 Detailed Description ............................... 435
9.19 chmboxes.c File Reference .................. 435
<table>
<thead>
<tr>
<th>Section</th>
<th>File Reference</th>
<th>Detailed Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>9.19.1</td>
<td>chmboxes.h</td>
<td>436</td>
</tr>
<tr>
<td>9.20</td>
<td>chmemcore.c</td>
<td>437</td>
</tr>
<tr>
<td>9.21</td>
<td>chmemheaps.c</td>
<td>439</td>
</tr>
<tr>
<td>9.22</td>
<td>chmemheaps.h</td>
<td>439</td>
</tr>
<tr>
<td>9.23</td>
<td>chmempools.c</td>
<td>440</td>
</tr>
<tr>
<td>9.24</td>
<td>chmempools.h</td>
<td>441</td>
</tr>
<tr>
<td>9.25</td>
<td>chmsg.c</td>
<td>443</td>
</tr>
<tr>
<td>9.26</td>
<td>chmsg.h</td>
<td>443</td>
</tr>
<tr>
<td>9.27</td>
<td>chmtx.c</td>
<td>444</td>
</tr>
<tr>
<td>9.28</td>
<td>chmtx.h</td>
<td>444</td>
</tr>
<tr>
<td>9.29</td>
<td>chobjfifos.h</td>
<td>445</td>
</tr>
<tr>
<td>9.30</td>
<td>chpipes.c</td>
<td>447</td>
</tr>
<tr>
<td>9.31</td>
<td>chpipes.h</td>
<td>447</td>
</tr>
<tr>
<td>9.32</td>
<td>chregistry.c</td>
<td>448</td>
</tr>
<tr>
<td>9.33</td>
<td>chregistry.h</td>
<td>449</td>
</tr>
<tr>
<td>9.34</td>
<td>chrestrictions.h</td>
<td>450</td>
</tr>
<tr>
<td>9.35</td>
<td>chschd.c</td>
<td>450</td>
</tr>
<tr>
<td>9.36</td>
<td>chschd.h</td>
<td>451</td>
</tr>
<tr>
<td>9.37</td>
<td>chsem.c</td>
<td>454</td>
</tr>
<tr>
<td>9.38</td>
<td>chsem.h</td>
<td>454</td>
</tr>
<tr>
<td>Section</td>
<td>File Reference</td>
<td>Detailed Description</td>
</tr>
<tr>
<td>-----------</td>
<td>----------------------</td>
<td>----------------------</td>
</tr>
<tr>
<td>9.39.1</td>
<td>Detailed Description</td>
<td>454</td>
</tr>
<tr>
<td>9.40</td>
<td>chsem.h File Reference</td>
<td>455</td>
</tr>
<tr>
<td>9.40.1</td>
<td>Detailed Description</td>
<td>456</td>
</tr>
<tr>
<td>9.41</td>
<td>chstats.c File Reference</td>
<td>456</td>
</tr>
<tr>
<td>9.41.1</td>
<td>Detailed Description</td>
<td>456</td>
</tr>
<tr>
<td>9.42</td>
<td>chstats.h File Reference</td>
<td>456</td>
</tr>
<tr>
<td>9.42.1</td>
<td>Detailed Description</td>
<td>457</td>
</tr>
<tr>
<td>9.43</td>
<td>chsys.c File Reference</td>
<td>457</td>
</tr>
<tr>
<td>9.43.1</td>
<td>Detailed Description</td>
<td>457</td>
</tr>
<tr>
<td>9.44</td>
<td>chsys.h File Reference</td>
<td>457</td>
</tr>
<tr>
<td>9.44.1</td>
<td>Detailed Description</td>
<td>459</td>
</tr>
<tr>
<td>9.45</td>
<td>chsystypes.h File Reference</td>
<td>459</td>
</tr>
<tr>
<td>9.45.1</td>
<td>Detailed Description</td>
<td>460</td>
</tr>
<tr>
<td>9.46</td>
<td>chthreads.c File Reference</td>
<td>460</td>
</tr>
<tr>
<td>9.46.1</td>
<td>Detailed Description</td>
<td>461</td>
</tr>
<tr>
<td>9.47</td>
<td>chthreads.h File Reference</td>
<td>461</td>
</tr>
<tr>
<td>9.47.1</td>
<td>Detailed Description</td>
<td>464</td>
</tr>
<tr>
<td>9.48</td>
<td>chtime.h File Reference</td>
<td>464</td>
</tr>
<tr>
<td>9.48.1</td>
<td>Detailed Description</td>
<td>465</td>
</tr>
<tr>
<td>9.49</td>
<td>chtm.c File Reference</td>
<td>465</td>
</tr>
<tr>
<td>9.49.1</td>
<td>Detailed Description</td>
<td>466</td>
</tr>
<tr>
<td>9.50</td>
<td>chtm.h File Reference</td>
<td>466</td>
</tr>
<tr>
<td>9.50.1</td>
<td>Detailed Description</td>
<td>466</td>
</tr>
<tr>
<td>9.51</td>
<td>chtrace.c File Reference</td>
<td>467</td>
</tr>
<tr>
<td>9.51.1</td>
<td>Detailed Description</td>
<td>467</td>
</tr>
<tr>
<td>9.52</td>
<td>chtrace.h File Reference</td>
<td>467</td>
</tr>
<tr>
<td>9.52.1</td>
<td>Detailed Description</td>
<td>468</td>
</tr>
<tr>
<td>9.53</td>
<td>chversion.h File Reference</td>
<td>469</td>
</tr>
<tr>
<td>9.53.1</td>
<td>Detailed Description</td>
<td>469</td>
</tr>
<tr>
<td>9.54</td>
<td>chvt.c File Reference</td>
<td>469</td>
</tr>
<tr>
<td>9.54.1</td>
<td>Detailed Description</td>
<td>469</td>
</tr>
<tr>
<td>9.55</td>
<td>chvt.h File Reference</td>
<td>469</td>
</tr>
<tr>
<td>9.55.1</td>
<td>Detailed Description</td>
<td>470</td>
</tr>
</tbody>
</table>

Index 471
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/RT Kernel Concepts

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

2.1 Naming Conventions

ChibiOS/RT APIs are all named following this convention: `ch<group><action><suffix>()`. The possible groups are: Sys, Sch, Time, VT, Thd, Sem, Mtx, Cond, Evt, Msg, Reg, SequentialStream, IO, IQ, OQ, Dbg, Core, Heap, Pool.

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.

- **"S"**, S-Class APIs are invokable only from the S-Locked state. See System States.

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

In ChibiOS/RT there are three logical interrupt classes:

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

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

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

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

2.4 System States

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

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

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

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

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

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

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

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

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

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

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

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

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:
2.4 System States

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

Any State except *  →  Fast IRQ  →  SFI

Fast IRQ return
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.

- **ABSPRIO**, absolute maximum software priority level, it can be higher than **HIGHPRIO** but the numerical values above **HIGHPRIO** up to **ABSPRIO** (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 .................................................. 19
- Release Information ..................................................... 20
- Customer Information .................................................. 22
- License Settings ......................................................... 23
- RT Kernel ................................................................. 25
- Version Numbers and Identification ................................. 26
- Configuration ............................................................. 29
  - Options ................................................................. 30
  - Checks ................................................................. 44
  - Restrictions .......................................................... 45
- Base Kernel Services .................................................... 46
  - Memory Alignment .................................................. 47
  - System Management ................................................ 49
  - Scheduler ............................................................. 68
  - Time and Intervals .................................................. 90
- Virtual Timers ......................................................... 101
- Threads ................................................................. 116
  - Time Measurement ................................................ 152
- Synchronization ......................................................... 155
  - Counting Semaphores ............................................. 156
  - Mutexes ............................................................... 171
  - Condition Variables .............................................. 183
  - Event Flags .......................................................... 194
  - Synchronous Messages ........................................... 215
- Dynamic Threads ....................................................... 220
- Registry ................................................................. 223
- Debug ................................................................. 230
  - Checks and Assertions ........................................... 231
  - Tracing ............................................................... 239
  - Statistics ............................................................ 247
- OS Library ............................................................... 250
  - Version Numbers and Identification ............................. 251
  - Synchronization .................................................... 252
    - Binary Semaphores .............................................. 253
    - Mailboxes ........................................................ 263
    - Pipes ............................................................... 278
  - Memory Management ............................................... 286
<table>
<thead>
<tr>
<th>Module</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Core Memory Manager</td>
<td>287</td>
</tr>
<tr>
<td>Memory Heaps</td>
<td>294</td>
</tr>
<tr>
<td>Memory Pools</td>
<td>300</td>
</tr>
<tr>
<td>Complex Services</td>
<td>319</td>
</tr>
<tr>
<td>Objects FIFOs</td>
<td>320</td>
</tr>
<tr>
<td>Dynamic Objects Factory</td>
<td>333</td>
</tr>
</tbody>
</table>
### Chapter 4

#### Hierarchical Index

##### 4.1 Class Hierarchy

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

<table>
<thead>
<tr>
<th>Class Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ch_dyn_element</td>
<td>352</td>
</tr>
<tr>
<td>ch_dyn_list</td>
<td>353</td>
</tr>
<tr>
<td>ch_dyn_mailbox</td>
<td>354</td>
</tr>
<tr>
<td>ch_dyn_object</td>
<td>356</td>
</tr>
<tr>
<td>ch_dyn_objects_fifo</td>
<td>357</td>
</tr>
<tr>
<td>ch_dyn_pipe</td>
<td>359</td>
</tr>
<tr>
<td>ch_dyn_semaphore</td>
<td>361</td>
</tr>
<tr>
<td>ch_mutex</td>
<td>363</td>
</tr>
<tr>
<td>ch_objects_factory</td>
<td>364</td>
</tr>
<tr>
<td>ch_objects_fifo</td>
<td>367</td>
</tr>
<tr>
<td>ch_registered_static_object</td>
<td>368</td>
</tr>
<tr>
<td>ch_semaphore</td>
<td>369</td>
</tr>
<tr>
<td>ch_binary_semaphore</td>
<td>351</td>
</tr>
<tr>
<td>ch_system</td>
<td>371</td>
</tr>
<tr>
<td>ch_system_debug</td>
<td>372</td>
</tr>
<tr>
<td>ch_thread</td>
<td>374</td>
</tr>
<tr>
<td>ch_threads_list</td>
<td>380</td>
</tr>
<tr>
<td>ch_threads_queue</td>
<td>382</td>
</tr>
<tr>
<td>ch_trace_buffer_t</td>
<td>383</td>
</tr>
<tr>
<td>ch_trace_event_t</td>
<td>385</td>
</tr>
<tr>
<td>ch_virtual_timers_list</td>
<td>391</td>
</tr>
<tr>
<td>ch_virtual_timer</td>
<td>389</td>
</tr>
<tr>
<td>chdebug_t</td>
<td>393</td>
</tr>
<tr>
<td>condition_variable</td>
<td>396</td>
</tr>
<tr>
<td>event_listener</td>
<td>397</td>
</tr>
<tr>
<td>event_source</td>
<td>398</td>
</tr>
<tr>
<td>guarded_memory_pool_t</td>
<td>400</td>
</tr>
<tr>
<td>heap_header</td>
<td>401</td>
</tr>
<tr>
<td>kernel_stats_t</td>
<td>402</td>
</tr>
<tr>
<td>mailbox_t</td>
<td>403</td>
</tr>
<tr>
<td>memcore_t</td>
<td>405</td>
</tr>
<tr>
<td>memory_heap</td>
<td>406</td>
</tr>
<tr>
<td>memory_pool_t</td>
<td>407</td>
</tr>
<tr>
<td>pipe_t</td>
<td>408</td>
</tr>
<tr>
<td>pool_header</td>
<td>411</td>
</tr>
<tr>
<td>thread_descriptor_t</td>
<td>412</td>
</tr>
<tr>
<td>time_measurement_t</td>
<td>413</td>
</tr>
</tbody>
</table>
tm_calibration_t ................................................................. 414
Chapter 5

Data Structure Index

5.1 Data Structures

Here are the data structures with brief descriptions:

- **ch_binary_semaphore**: Binary semaphore type .................................................. 351
- **ch_dyn_element**: Type of a dynamic object list element ................................... 352
- **ch_dyn_list**: Type of a dynamic object list ....................................................... 353
- **ch_dyn_mailbox**: Type of a dynamic buffer object ........................................... 354
- **ch_dyn_object**: Type of a dynamic buffer object ............................................. 356
- **ch_dyn_objects_fifo**: Type of a dynamic buffer object ..................................... 357
- **ch_dyn_pipe**: Type of a dynamic pipe object .................................................. 359
- **ch_dyn_semaphore**: Type of a dynamic semaphore ......................................... 361
- **ch_mutex**: Mutex structure .............................................................................. 363
- **ch_objects_factory**: Type of the factory main object ....................................... 364
- **ch_objects_fifo**: Type of an objects FIFO ....................................................... 367
- **ch_registered_static_object**: Type of a registered object ................................. 368
- **ch_semaphore**: Semaphore structure .............................................................. 369
- **ch_system**: System data structure ................................................................... 371
- **ch_system_debug**: System debug data structure ............................................. 372
- **ch_thread**: Structure representing a thread .................................................... 374
- **ch_threads_list**: Generic threads single link list, it works like a stack .............. 380
- **ch_threads_queue**: Generic threads bidirectional linked list header and element .. 382
- **ch_trace_buffer_t**: Trace buffer header .......................................................... 383
<table>
<thead>
<tr>
<th>Data Structure Index</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ch_trace_event_t</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>
<tr>
<td><strong>thread_descriptor_t</strong></td>
</tr>
<tr>
<td><strong>time_measurement_t</strong></td>
</tr>
<tr>
<td><strong>tm_calibration_t</strong></td>
</tr>
</tbody>
</table>
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 ........................................ 417
- `chalign.h` Memory alignment macros and structures ......................... 418
- `chbsem.h` Binary semaphores structures and macros ......................... 418
- `chchecks.h` Configuration file checks header ................................ 420
- `chcond.c` Condition Variables code ........................................... 420
- `chcond.h` Condition Variables macros and structures ....................... 421
- `chconf.h` Configuration file template ........................................... 422
- `chcustomer.h` Customer-related info ........................................... 424
- `chdebug.c` Debug support code .................................................. 425
- `chdebug.h` Debug support macros and structures ................................ 426
- `chdynamic.c` Dynamic threads code ............................................. 426
- `chdynamic.h` Dynamic threads macros and structures ....................... 426
- `chevents.c` Events code ........................................................ 427
- `chevents.h` Events macros and structures ....................................... 428
- `chfactory.c` ChibiOS objects factory and registry code .................... 430
- `chfactory.h` ChibiOS objects factory structures and macros ............... 431
- `chlib.h` ChibiOS/LIB main include file ........................................ 434
- `chlicense.h` License Module macros and structures ........................... 435
- `chmboxes.c` Mailboxes code .................................................... 435
<table>
<thead>
<tr>
<th>File Name</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>chmboxes.h</td>
<td>Mailboxes macros and structures</td>
<td>436</td>
</tr>
<tr>
<td>chmemcore.c</td>
<td>Core memory manager code</td>
<td>437</td>
</tr>
<tr>
<td>chmemcore.h</td>
<td>Core memory manager macros and structures</td>
<td>438</td>
</tr>
<tr>
<td>chmemheaps.c</td>
<td>Memory heaps code</td>
<td>439</td>
</tr>
<tr>
<td>chmemheaps.h</td>
<td>Memory heaps macros and structures</td>
<td>439</td>
</tr>
<tr>
<td>chmempools.c</td>
<td>Memory Pools code</td>
<td>440</td>
</tr>
<tr>
<td>chmempools.h</td>
<td>Memory Pools macros and structures</td>
<td>441</td>
</tr>
<tr>
<td>chmsg.c</td>
<td>Messages code</td>
<td>443</td>
</tr>
<tr>
<td>chmsg.h</td>
<td>Messages macros and structures</td>
<td>443</td>
</tr>
<tr>
<td>chmtx.c</td>
<td>Mutexes code</td>
<td>444</td>
</tr>
<tr>
<td>chmtx.h</td>
<td>Mutexes macros and structures</td>
<td>444</td>
</tr>
<tr>
<td>chobjectos.h</td>
<td>Objects FIFO structures and macros</td>
<td>445</td>
</tr>
<tr>
<td>chpipes.c</td>
<td>Pipes code</td>
<td>447</td>
</tr>
<tr>
<td>chpipes.h</td>
<td>Pipes macros and structures</td>
<td>447</td>
</tr>
<tr>
<td>chregistry.c</td>
<td>Threads registry code</td>
<td>447</td>
</tr>
<tr>
<td>chregistry.h</td>
<td>Threads registry macros and structures</td>
<td>448</td>
</tr>
<tr>
<td>chrestrictions.h</td>
<td>Licensing restrictions header</td>
<td>450</td>
</tr>
<tr>
<td>chschd.c</td>
<td>Scheduler code</td>
<td>450</td>
</tr>
<tr>
<td>chschd.h</td>
<td>Scheduler macros and structures</td>
<td>451</td>
</tr>
<tr>
<td>chsem.c</td>
<td>Semaphores code</td>
<td>454</td>
</tr>
<tr>
<td>chsem.h</td>
<td>Semaphores macros and structures</td>
<td>455</td>
</tr>
<tr>
<td>chstats.c</td>
<td>Statistics module code</td>
<td>456</td>
</tr>
<tr>
<td>chstats.h</td>
<td>Statistics module macros and structures</td>
<td>456</td>
</tr>
<tr>
<td>chsys.c</td>
<td>System related code</td>
<td>457</td>
</tr>
<tr>
<td>chsys.h</td>
<td>System related macros and structures</td>
<td>457</td>
</tr>
<tr>
<td>chsystypes.h</td>
<td>System types header</td>
<td>459</td>
</tr>
<tr>
<td>chthreads.c</td>
<td>Threads code</td>
<td>460</td>
</tr>
<tr>
<td>chthreads.h</td>
<td>Threads module macros and structures</td>
<td>461</td>
</tr>
<tr>
<td>chtime.h</td>
<td>Time and intervals macros and structures</td>
<td>464</td>
</tr>
</tbody>
</table>
6.1 File List

chtm.c  Time Measurement module code ............................................. 465
chtm.h  Time Measurement module macros and structures ....................... 466
chtrace.c  Tracer code ................................................................. 467
chtrace.h  Tracer macros and structures ............................................. 467
chversion.h  Version Module macros and structures ............................... 469
chvt.c  Time and Virtual Timers module code .................................... 469
chvt.h  Time and Virtual Timers module macros and structures ................ 469
Chapter 7

Module Documentation

7.1 Release and Licensing

7.1.1 Detailed Description

Modules

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

7.2.1 Detailed Description

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

Macros

- `#define _CHIBIOS_`  
  ChibiOS product identification macro.
- `#define CH_VERSION_STABLE 1`  
  Stable release flag.

ChibiOS version identification

- `#define CH_VERSION "19.1.3"`  
  ChibiOS version string.
- `#define CH_VERSION_YEAR 19`  
  ChibiOS version release year.
- `#define CH_VERSION_MONTH 1`  
  ChibiOS version release month.
- `#define CH_VERSION_PATCH 3`  
  ChibiOS version patch number.
- `#define CH_VERSION_NICKNAME "Maiori"`  
  ChibiOS version nickname.

7.2.2 Macro Definition Documentation

7.2.2.1 `#define _CHIBIOS_`

ChibiOS product identification macro.

7.2.2.2 `#define CH_VERSION_STABLE 1`

Stable release flag.

7.2.2.3 `#define CH_VERSION "19.1.3"`

ChibiOS version string.

7.2.2.4 `#define CH_VERSION_YEAR 19`

ChibiOS version release year.

7.2.2.5 `#define CH_VERSION_MONTH 1`

ChibiOS version release month.
7.2 Release Information

7.2.2.6  \#define CH_VERSION_PATCH 3

ChibiOS version patch number.

7.2.2.7  \#define CH_VERSION_NICKNAME "Maiori"

ChibiOS version nickname.
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_LICENSE CH_LICENSE_GPL
  
  Current license.

Licensed Products

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

7.3.2 Macro Definition Documentation

7.3.2.1 `#define CH_CUSTOMER_ID_STRING "Santa, North Pole"

Customer readable identifier.

7.3.2.2 `#define CH_CUSTOMER_ID_CODE "xxxx-yyyy"

Customer code.

7.3.2.3 `#define CH_LICENSE CH_LICENSE_GPL

Current license.

Note

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

7.4.1 Detailed Description

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

Macros

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

Allowed Features Levels

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

Deployment Options

- \#define CH_DEPLOY_UNLIMITED -1
- \#define CH_DEPLOY_NONE 0

Licensing Options

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

7.4.2 Macro Definition Documentation

7.4.2.1 \#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 #define CH_LICENSE_ID_STRING "N/A"
Customer identification string.
This information is only available for registered commercial users.

7.4.2.3 #define CH_LICENSE_ID_CODE "N/A"
Customer code.
This information is only available for registered commercial users.

7.4.2.4 #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 #define CH_LICENSE_FEATURES CH_FEATURES_FULL
Code functionality restrictions.

7.4.2.6 #define CH_LICENSE_MAX_DEPLOY CH_DEPLOY_UNLIMITED
Code deploy restrictions.
This is the per-core deploy limit allowed under the current license scheme.
7.5 RT Kernel

7.5.1 Detailed Description

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

Modules

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

7.6.1  Detailed Description

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

Kernel related info.

Macros

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

ChibiOS/RT version identification

- `#define CH_KERNEL_VERSION "6.0.4"`
  Kernel version string.
- `#define CHKERNEL_MAJOR 6`
  Kernel version major number.
- `#define CHKERNEL_MINOR 0`
  Kernel version minor number.
- `#define CHKERNEL_PATCH 4`
  Kernel version patch number.

Constants for configuration options

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

Functions

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

7.6.2  Macro Definition Documentation

7.6.2.1  `#define _CHIBIOS_RT_`

ChibiOS/RT identification macro.

7.6.2.2  `#define CH_KERNEL_STABLE 1`

Stable release flag.
7.6.2.3 #define CH_KERNEL_VERSION "6.0.4"
Kernel version string.

7.6.2.4 #define CH_KERNEL_MAJOR 6
Kernel version major number.

7.6.2.5 #define CH_KERNEL_MINOR 0
Kernel version minor number.

7.6.2.6 #define CH_KERNEL_PATCH 4
Kernel version patch number.

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

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

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

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

7.6.3 Function Documentation

7.6.3.1 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

|   | reason | pointer to an error string |

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

```
chSysHalt -> _trace_halt -> trace_next
```
7.7  Configuration

7.7.1  Detailed Description

Modules

- Options
- Checks
- Restrictions
7.8 Options

7.8.1 Detailed Description

Kernel related settings and hooks.

System timers settings

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

Kernel parameters and options

- `#define CH_CFG_TIME_QUANTUM 0`
  Round robin interval.
- `#define CH_CFG_MEMCORE_SIZE 0`
  Managed RAM size.
- `#define CH_CFG_NO_IDLE_THREAD FALSE`
  Idle thread automatic spawn suppression.

Performance options

- `#define CH_CFG_OPTIMIZE_SPEED TRUE`
  OS optimization.

Subsystem options

- `#define CH_CFG_USE_TM TRUE`
  Time Measurement APIs.
- `#define CH_CFG_USE_REGISTRY TRUE`
  Threads registry APIs.
- `#define CH_CFG_USE_WAITEXIT TRUE`
  Threads synchronization APIs.
- `#define CH_CFG_USE_SEMAPHORES TRUE`
  Semaphores APIs.
- `#define CH_CFG_USE_SEMAPHORES_PRIORITY FALSE`
  Semaphores queuing mode.
- `#define CH_CFG_USE_MUTEXES TRUE`
  Mutexes APIs.
- `#define CH_CFG_USE_MUTEXES_RECURSIVE FALSE`
  Enables recursive behavior on mutexes.
- `#define CH_CFG_USE_CONDVARS TRUE`
Conditional Variables APIs.
• #define CH_CFG_USE_CONDVARS_TIMEOUT TRUE
  Conditional Variables APIs with timeout.
• #define CH_CFG_USE_EVENTS TRUE
  Events Flags APIs.
• #define CH_CFG_USE_EVENTS_TIMEOUT TRUE
  Events Flags APIs with timeout.
• #define CH_CFG_USE_MESSAGES TRUE
  Synchronous Messages APIs.
• #define CH_CFG_USE_MESSAGES_PRIORITY FALSE
  Synchronous Messages queuing mode.
• #define CH_CFG_USE_MAILBOXES TRUE
  Mailboxes APIs.
• #define CH_CFG_USE_MEMCORE TRUE
  Core Memory Manager APIs.
• #define CH_CFG_USE_HEAP TRUE
  Heap Allocator APIs.
• #define CH_CFG_USE_MEMPOOLS TRUE
  Memory Pools Allocator APIs.
• #define CH_CFG_USE_OBJ_FIFOS TRUE
  Objects FIFOs APIs.
• #define CH_CFG_USE_PIPES TRUE
  Pipes APIs.
• #define CH_CFG_USE_DYNAMIC TRUE
  Dynamic Threads APIs.

Objects factory options

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

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

Kernel hooks

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

7.8.2 Macro Definition Documentation

7.8.2.1 #define CH_CFG_ST_RESOLUTION 32

System time counter resolution.

Note

Allowed values are 16 or 32 bits.

7.8.2.2 #define CH_CFG_ST_FREQUENCY 10000

System tick frequency.

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

7.8.2.3 #define CH_CFG_INTERVALS_SIZE 32

Time intervals data size.

Note

Allowed values are 16, 32 or 64 bits.

7.8.2.4 #define CH_CFG_TIME_TYPES_SIZE 32

Time types data size.

Note

Allowed values are 16 or 32 bits.

7.8.2.5 #define CH_CFG_ST_TIMEDELTA 2

Time delta constant for the tick-less mode.

Note

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

7.8.2.6 #define CH_CFG_TIME_QUANTUM 0

Round robin interval.

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

Note

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

The round robin preemption is not supported in tickless mode and must be set to zero in that case.
7.8.2.7 #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.

7.8.2.8 #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 #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 #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 #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.

7.8.2.12 #define CH_CFG_USE_WAITEXIT TRUE

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

Note
The default is TRUE.

7.8.2.13 #define CH_CFG_USE_SEMAPHORES TRUE

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

Note
The default is TRUE.

7.8.2.14 #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.

7.8.2.15 #define CH_CFG_USE_MUTEXES TRUE

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

Note
The default is TRUE.

7.8.2.16 #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.

7.8.2.17 #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.
7.8.2.18  #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.

7.8.2.19  #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.

7.8.2.20  #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.

7.8.2.21  #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.

7.8.2.22  #define CH_CFG_USE_MESSAGES_PRIORITY FALSE

Synchronous Messages queuing mode.
If enabled then messages are served by priority rather than in FIFO order.

Note

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

7.8.2.23  #define CH_CFG_USE_MAILBOXES TRUE

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

Note

The default is TRUE.
Requires CH_CFG_USE_SEMAPHORES.

7.8.2.24  #define CH_CFG_USE_MEMCORE TRUE

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

Note

The default is TRUE.

7.8.2.25  #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.

7.8.2.26  #define CH_CFG_USE_MEMPOOLS TRUE

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

Note

The default is TRUE.

7.8.2.27  #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.
7.8.2.28 #define CH_CFG_USE_PIPES TRUE
Pipes APIs.
If enabled then the pipes APIs are included in the kernel.

Note
The default is TRUE.

7.8.2.29 #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.

7.8.2.30 #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.

7.8.2.31 #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.

7.8.2.32 #define CH_CFG_FACTORY_OBJECTS_REGISTRY TRUE
Enables the registry of generic objects.

7.8.2.33 #define CH_CFG_FACTORY_GENERIC_BUFFERS TRUE
Enables factory for generic buffers.

7.8.2.34 #define CH_CFG_FACTORY_SEMAPHORES TRUE
Enables factory for semaphores.

7.8.2.35 #define CH_CFG_FACTORY_MAILBOXES TRUE
Enables factory for mailboxes.
7.8.2.36  #define CH_CFG_FACTORY_OBJ_FIFOS TRUE
Enables factory for objects FIFOs.

7.8.2.37  #define CH_CFG_FACTORY_PIPES TRUE
Enables factory for Pipes.

7.8.2.38  #define CH_DBG_STATISTICS FALSE
Debug option, kernel statistics.

Note
The default is FALSE.

7.8.2.39  #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.

7.8.2.40  #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.

7.8.2.41  #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.

7.8.2.42  #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.
7.8.2.43  #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.

7.8.2.44  #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.

7.8.2.45  #define CH_DBG_FILL_THREADS TRUE
Debug option, stacks initialization.
If enabled then the threads working area is filled with a byte value when a thread is created. This can be useful for
the runtime measurement of the used stack.

Note
The default is FALSE.

7.8.2.46  #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.

7.8.2.47  #define CH_CFG_SYSTEM_EXTRA_FIELDS /* Add threads custom fields here.*/
System structure extension.
User fields added to the end of the ch_system_t structure.

7.8.2.48  #define CH_CFG_SYSTEM_INIT_HOOK( )
Value:

\{
  /* Add threads initialization code here. */
\}
System initialization hook.
User initialization code added to the \texttt{chSysInit()} function just before interrupts are enabled globally.

7.8.2.49 \#define CH_CFG_THREAD_EXTRA_FIELDS /* Add threads custom fields here.*/

Threads descriptor structure extension.
User fields added to the end of the \texttt{thread_t} structure.

7.8.2.50 \#define CH_CFG_THREAD_INIT_HOOK( \textit{tp} )

\textbf{Value:}
\begin{verbatim}
     
     /* Add threads initialization code here.*/

     
\end{verbatim}

Threads initialization hook.
User initialization code added to the \_\texttt{thread_init()} function.

\textbf{Note}
It is invoked from within \_\texttt{thread_init()} and implicitly from all the threads creation APIs.

7.8.2.51 \#define CH_CFG_THREAD_EXIT_HOOK( \textit{tp} )

\textbf{Value:}
\begin{verbatim}
     
     /* Add threads finalization code here.*/

     
\end{verbatim}

Threads finalization hook.
User finalization code added to the \texttt{chThdExit()} API.

7.8.2.52 \#define CH_CFG_CONTEXT_SWITCH_HOOK( \textit{ntp}, \textit{otp} )

\textbf{Value:}
\begin{verbatim}
     
     /* Context switch code here.*/

     
\end{verbatim}

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

7.8.2.53 \#define CH_CFG_IRQ_PROLOGUE_HOOK( )

\textbf{Value:}
\begin{verbatim}
     
     /* IRQ prologue code here.*/

     
\end{verbatim}

ISR enter hook.
7.8.2.54  #define CH_CFG_IRQ_EPILOGUE_HOOK( )

Value:

/* IRQ epilogue code here.*/
ISR exit hook.

7.8.2.55  #define CH_CFG_IDLE_ENTER_HOOK( )

Value:

/* Idle-enter code here.*/
Idle thread enter hook.

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

7.8.2.56  #define CH_CFG_IDLE_LEAVE_HOOK( )

Value:

/* Idle-leave code here.*/
Idle thread leave hook.

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

7.8.2.57  #define CH_CFG_IDLE_LOOP_HOOK( )

Value:

/* Idle loop code here.*/
Idle Loop hook.
This hook is continuously invoked by the idle thread loop.

7.8.2.58  #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 Options

7.8.2.59  

#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.

7.8.2.60  

#define CH_CFG_TRACE_HOOK( tep )

Value:

/* Trace code here. */

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

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

7.10.1 Detailed Description

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

7.11.1 Detailed Description

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

Modules

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

7.12.1 Detailed Description

Memory Alignment services.

Memory alignment support macros

- **#define MEM_ALIGN_MASK(a)** \((\text{size}_t)(a) - 1U)\)
  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_ISAligned(p, a)** \(((\text{size}_t)(p) & MEM_ALIGN_MASK(a)) == 0U)\)
  Returns whatever a pointer or memory size is aligned.

- **#define MEM_IsVALID_ALIGNMENT(a)** \(((\text{size}_t)(a) != 0U) && ((\text{size}_t)(a) & ((\text{size}_t)(a) - 1U)) == 0U)\)
  Returns whatever a constant is a valid alignment.

7.12.2 Macro Definition Documentation

7.12.2.1 **#define MEM_ALIGN_MASK(a)** \((\text{size}_t)(a) - 1U)\)

Alignment mask constant.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>a</th>
<th>alignment, must be a power of two</th>
</tr>
</thead>
</table>

7.12.2.2 **#define MEM_ALIGN_PREV(p, a)**

Value:

```c
/*lint -save -e9033 [10.8] The cast is safe.*/

 MEM_ALIGN_PREV((size_t)(p) & ~MEM_ALIGN_MASK(a))

/*lint -restore*/
```

Aligns to the previous aligned memory address.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>p</th>
<th>variable to be aligned</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>in</th>
<th>a</th>
<th>alignment, must be a power of two</th>
</tr>
</thead>
</table>

7.12.2.3 **#define MEM_ALIGN_NEXT(p, a)**

Value:

```c
/*lint -save -e9033 [10.8] The cast is safe.*/

 MEM_ALIGN_PREV((size_t)(p) & ~MEM_ALIGN_MASK(a)) + MEM_ALIGN_MASK(a)

/*lint -restore*/
```
Aligns to the next aligned memory address.

Parameters

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

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

Returns whatever a pointer or memory size is aligned.

Parameters

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

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

Returns whatever a constant is a valid alignment.

Valid alignments are powers of two.

Parameters

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

7.13.1 Detailed Description

System related APIs and services:

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

Macros

- `#define chSysGetRealtimeCounterX() (rtcnt_t)port_rt_get_counter_value()`
  
  Returns the current value of the system real time counter.

- `#define chSysSwitch(ntp, otp)`
  
  Performs a context switch.

Masks of executable integrity checks.

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

ISRs abstraction macros

- `#define CH_IRQ_IS_VALID_PRIORITY(prio) PORT_IRQ_IS_VALID_PRIORITY(prio)`
  
  Priority level validation macro.

- `#define CH_IRQ_IS_VALID_KERNEL_PRIORITY(prio) PORT_IRQ_IS_VALID_KERNEL_PRIORITY(prio)`
  
  Priority level validation macro.

- `#define CH_IRQ_PROLOGUE()`
  
  IRQ handler enter code.

- `#define CH_IRQ_EPILOGUE()`
  
  IRQ handler exit code.

- `#define CH_IRQ_HANDLER(id) PORT_IRQ_HANDLER(id)`
  
  Standard normal IRQ handler declaration.

Fast ISRs abstraction macros

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

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

Functions

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

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

7.13.2 Macro Definition Documentation

7.13.2.1 #define CH_IRQ_IS_VALID_PRIORITY( prio ) PORT_IRQ_IS_VALID_PRIORITY(prio)

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

Parameters

| in | prio | the priority level |

Returns

Priority range result.

Return values

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

7.13.2.2 #define CH_IRQ_IS_VALID_KERNEL_PRIORITY( prio ) PORT_IRQ_IS_VALID_KERNEL_PRIORITY(prio)

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

Parameters

| in | prio | the priority level |

Returns

Priority range result.

Return values

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

7.13.2.3 #define CH_IRQ_PROLOGUE( )

Value:

ChibiOS/RT
PORT_IRQ_PROLOGUE();
CH_CFG_IRQ_PROLOGUE_HOOK();
_stats_increase_irq();
_trace_isr_enter(__func__);
_dbg_check_enter_isr();

IRQ handler enter code.

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

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

7.13.2.4#define CH_IRQ_EPILOGUE( )

Value:

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

IRQ handler exit code.

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

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

7.13.2.5#define CH_IRQ_HANDLER( id ) PORT_IRQ_HANDLER(id)

Standard normal IRQ handler declaration.

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

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

7.13.2.6#define CH_FAST_IRQ_HANDLER( id ) PORT_FAST_IRQ_HANDLER(id)

Standard fast IRQ handler declaration.
Note

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

Function Class:

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

7.13.2.7  
#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

| in | freq | clock frequency, in Hz, of the realtime counter |
| in | sec  | number of seconds |

Returns
The number of cycles.

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

7.13.2.8  
#define MS2RTC(  freq,  msec ) (rtcnt_t)(((freq) + 999UL) / 1000UL) ∗ (msec))

Milliseconds to realtime counter.
Converts from milliseconds to realtime counter cycles.

Note
The result is rounded upward to the next millisecond boundary.
The macro assumes that freq >= 1000.

Parameters

| in | freq | clock frequency, in Hz, of the realtime counter |
| in | msec | number of milliseconds |

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.
Module Documentation

7.13.2.9  #define US2RTC( freq, usec ) (rtcnt_t)(((freq) + 999999UL) / 1000000UL) + (usec))

Microseconds to realtime counter.
Converts from microseconds to realtime counter cycles.

Note
The result is rounded upward to the next microsecond boundary.
The macro assumes that \( \text{freq} \geq 1000000 \).

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.

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

Realtime counter cycles to seconds.
Converts from realtime counter cycles number to seconds.

Note
The result is rounded up to the next second boundary.
The macro assumes that \( \text{freq} \geq 1 \).

Parameters

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

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

Realtime counter cycles to milliseconds.
Converts from realtime counter cycles number to milliseconds.
7.13 System Management

Note

The result is rounded up to the next millisecond boundary.
The macro assumes that \( \text{freq} \geq 1000 \).

Parameters

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

Returns

The number of milliseconds.

Function Class:

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

7.13.2.12

\[
define \text{RTC2US}( \text{freq, n} ) (((\text{n} - 1UL) / ((\text{freq}) / 1000000UL)) + 1UL)
\]

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 \( \text{freq} \geq 1000000 \).

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>n</td>
<td>number of cycles</td>
</tr>
</tbody>
</table>

Returns

The number of microseconds.

Function Class:

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

7.13.2.13

\[
define \text{chSysGetRealtimeCounterX}( ) (\text{rtcnt_t}) \text{port}\_\text{rt}\_\text{get}\_\text{counter}\_\text{value}()
\]

Returns the current value of the system real time counter.

Note

This function is only available if the port layer supports the option \text{PORT\_SUPPORTS\_RT}.

Returns

The value of the system realtime counter of type \text{rtcnt_t}.

Function Class:

This is an \text{X-Class} API, this function can be invoked from any context.
7.13.2.14 #define chSysSwitch( ntp, otp )

Value:

\[
\begin{array}{c}
_\text{trace}\_\text{switch}(\text{ntp}, \text{otp}); \\
_\text{stats}\_\text{ctxswc}(\text{ntp}, \text{otp}); \\
\text{CH}\_\text{CFG}\_\text{CONTEXT}\_\text{SWITCH}\_\text{HOOK}(\text{ntp}, \text{otp}); \\
\text{port}\_\text{switch}(\text{ntp}, \text{otp});
\end{array}
\]

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.

7.13.3 Function Documentation

7.13.3.1 THD_WORKING_AREA ( ch_idle_thread_wa , PORT_IDLE_THREAD_STACK_SIZE )
Idle thread working area.

7.13.3.2 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 |

7.13.3.3 void chSysInit ( void )
ChibiOS/RT initialization.
After executing this function the current instructions stream becomes the main thread.

Precondition
Interrupts must disabled before invoking this function.
Postcondition
The main thread is created with priority NORMALPRIO and interrupts are enabled.

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

Here is the call graph for this function:

7.13.3.4 void chSysHalt ( const char *reason )

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

Can be invoked from any system state.

Parameters

| in | reason | pointer to an error string |

Function Class:

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

Here is the call graph for this function:

```
chSysHalt → _trace_halt → trace_next
```

7.13.3.5 bool chSysIntegrityCheckI ( unsigned testmask )

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

Note

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

Parameters

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

Returns

The test result.

Return values

| false | The test succeeded. |
| true  | Test failed. |
Function Class:

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

Here is the call graph for this function:

```
chSysIntegrityCheckI ← chDbgCheckClassI ← chSysHalt
```

### 7.13.3.6 void chSysTimerHandler ( 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:

```
chSysTimerHandlerI ← chDbgCheckClassI ← chVTDoTickI ← chSysHalt
chSysUnlockFromISR ← chSysLockFromISR
chVTGetSystemTimeX ← chTimeDiffX ← chTimeAddX
```

### 7.13.3.7 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
  |   ...
  |   chSysLockFromISR
  |   ...
  |   chSysLock
  |   ...
  |   _stats_start_measure
  |   _crit_isr
  |   ...
  |   _dbg_check_lock_from_isr
  |   ...
  |   _stats_start_measure
  |   _crit_thd
  |   ...
  |   _dbg_check_lock
```

### 7.13.3.8 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.
7.13.3.9  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

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

Function Class:

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

7.13.3.10  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 \texttt{PORT SUPPORTS RT}.

Parameters

| in  | cycles | number of cycles |

Function Class:

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

Here is the call graph for this function:

![Call Graph]

7.13.3.11 \textbf{static void chSysDisable ( void )} [inline],[static]

Raises the system interrupt priority mask to the maximum level.
All the maskable interrupt sources are disabled regardless their hardware priority.

Note

Do not invoke this API from within a kernel lock.

Function Class:

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

Here is the call graph for this function:

![Call Graph]

7.13.3.12 \textbf{static void chSysSuspend ( void )} [inline],[static]

Raises the system interrupt priority mask to system level.
The interrupt sources that should not be able to preempt the kernel are disabled, interrupt sources with higher priority are still enabled.
Note
Do not invoke this API from within a kernel lock.
This API is no replacement for chSysLock(), the chSysLock() could do more than just disable the interrupts.

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

Here is the call graph for this function:

```
chSysSuspend -> _dbg_check_suspend -> chSysHalt
```

7.13.3.13 static void chSysEnable ( void ) [inline],[static]

Lowers the system interrupt priority mask to user level.
All the interrupt sources are enabled.

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

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

Here is the call graph for this function:

```
chSysEnable -> _dbg_check_enable -> chSysHalt
```

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

Enters the kernel lock state.

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

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

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

Here is the call graph for this function:

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

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

It is good practice to invoke this API before invoking any I-class syscall from an interrupt handler.
This API must be invoked exclusively from interrupt handlers.

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

Here is the call graph for this function:

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

Leaves the kernel lock state from within an interrupt handler.

Note

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

Function Class:

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

Here is the call graph for this function:

7.13.3.18 static void chSysUnconditionalLock ( void ) [inline],[static]

Unconditionally enters the kernel lock state.

Note

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

Function Class:

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

```
chSysUnconditionalLock  chSysLock
    `___stats_start_measure
    `___crit_thd
    `___dbg_check_lock
```

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

Unconditionally leaves the kernel lock state.

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

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

Here is the call graph for this function:

```
chSysUnconditionalUnlock  chSysUnlock
    `___dbg_check_unlock
    `___stats_stop_measure
    `___crit_thd
```

7.13.3.20  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

Point to the idle thread.

Function Class:

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

7.14.1 Detailed Description

This module provides the default portable scheduler code.

Macros

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

Wakeup status codes

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

Priority constants

- `#define NOPRIOR (tprio_t)0
  Ready list header priority.
- `#define IDLEPRIOR (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_QUEUE (tstate_t)4
  On a queue.
- `#define CH_STATE_WTSEM (tstate_t)5
  ChibiOS/RT
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.

Typedefs
- typedef struct ch_thread thread_t
  Type of a thread structure.
- typedef thread_t * thread_reference_t
  Type of a thread reference.
- typedef struct ch_threads_list threads_list_t
  Type of a generic threads single link list, it works like a stack.
- typedef struct ch_threads_queue threads_queue_t
  Type of a generic threads bidirectional linked list header and element.
- typedef struct ch_ready_list ready_list_t
  Type of a ready list header.
- typedef void(* vtfunc_t) (void *p)
Type of a Virtual Timer callback function.

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

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

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

- typedef struct ch_system ch_system_t
  Type of system data structure.

Data Structures

- struct ch_threads_list
  Generic threads single link list, it works like a stack.

- struct ch_threads_queue
  Generic threads bidirectional linked list header and element.

- struct ch_thread
  Structure representing a thread.

- struct ch_virtual_timer
  Virtual Timer descriptor structure.

- struct ch_virtual_timers_list
  Virtual timers list header.

- struct ch_system_debug
  System debug data structure.

- struct ch_system
  System data structure.

Functions

- void _scheduler_init (void)
  Scheduler initialization.

- void queue_prio_insert (thread_t *tp, threads_queue_t *tqp)
  Inserts a thread into a priority ordered queue.

- void queue_insert (thread_t *tp, threads_queue_t *tqp)
  Inserts a thread into a queue.

- thread_t * queue_fifo_remove (threads_queue_t *tqp)
  Removes the first-out thread from a queue and returns it.

- thread_t * queue_lifo_remove (threads_queue_t *tqp)
  Removes the last-out thread from a queue and returns it.

- thread_t * queue_dequeue (thread_t *tp)
  Removes a thread from a queue and returns it.

- void list_insert (thread_t *tp, threads_list_t *tlp)
  Pushes a thread_t on top of a stack list.

- thread_t * list_remove (threads_list_t *tlp)
  Pops a thread from the top of a stack list and returns it.

- thread_t * chSchReadyI (thread_t *tp)
  Inserts a thread in the Ready List placing it behind its peers.

- thread_t * chSchReadyAheadI (thread_t *tp)
  Inserts a thread in the Ready List placing it ahead its peers.

- void chSchGoSleepS (tstate_t newstate)
7.14 Scheduler

Puts the current thread to sleep into the specified state.

- `msg_t chSchGoSleepTimeoutS (tstate_t newstate, sysinterval_t timeout)`
  Puts the current thread to sleep into the specified state with timeout specification.

- `void chSchWakeupS (thread_t *ntp, msg_t msg)`
  Wakes up a thread.

- `void chSchRescheduleS (void)`
  Performs a reschedule if a higher priority thread is runnable.

- `bool chSchIsPreemptionRequired (void)`
  Evaluates if preemption is required.

- `void chSchDoRescheduleBehind (void)`
  Switches to the first thread on the runnable queue.

- `void chSchDoRescheduleAhead (void)`
  Switches to the first thread on the runnable queue.

- `void chSchDoReschedule (void)`
  Switches to the first thread on the runnable queue.

- `static void list_init (threads_list_t *tlp)`
  Threads list initialization.

- `static bool list_isempty (threads_list_t *tlp)`
  Evaluates to true if the specified threads list is empty.

- `static bool list_notempty (threads_list_t *tlp)`
  Evaluates to true if the specified threads list is not empty.

- `static void queue_init (threads_queue_t *tqp)`
  Threads queue initialization.

- `static bool queue_isempty (const threads_queue_t *tqp)`
  Evaluates to true if the specified threads queue is empty.

- `static bool queue_notempty (const threads_queue_t *tqp)`
  Evaluates to true if the specified threads queue is not empty.

- `static bool chSchIsRescRequired (void)`
  Determines if the current thread must reschedule.

- `static bool chSchCanYieldS (void)`
  Determines if yielding is possible.

- `static void chSchDoYieldS (void)`
  Yields the time slot.

- `static void chSchPreemption (void)`
  Inline-able preemption code.

Variables

- `ch_system_t ch`
  System data structures.

7.14.2 Macro Definition Documentation

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

7.14.2.2 `#define MSG_TIMEOUT (msg_t)-1`
Wakeup caused by a timeout condition.
7.14.2.3  #define MSG_RESET (msg_t)-2
Wakeup caused by a reset condition.

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

7.14.2.5  #define IDLEPRIO (tprio_t)1
Idle priority.

7.14.2.6  #define LOWPRIO (tprio_t)2
Lowest priority.

7.14.2.7  #define NORMALPRIO (tprio_t)128
Normal priority.

7.14.2.8  #define HIGHPRIO (tprio_t)255
Highest priority.

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

7.14.2.10 #define CH_STATE_CURRENT (tstate_t)1
Currently running.

7.14.2.11 #define CH_STATE_WTSTART (tstate_t)2
Just created.

7.14.2.12 #define CH_STATE_SUSPENDED (tstate_t)3
Suspended state.

7.14.2.13 #define CH_STATE_QUEUED (tstate_t)4
On a queue.

7.14.2.14 #define CH_STATE_WTSEM (tstate_t)5
On a semaphore.
7.14.2.15 #define CH_STATE_WTMTX (tstate_t)6
On a mutex.

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

7.14.2.17 #define CH_STATE_SLEEPING (tstate_t)8
Sleeping.

7.14.2.18 #define CH_STATE_WTEXIT (tstate_t)9
Waiting a thread.

7.14.2.19 #define CH_STATE_WTOREVT (tstate_t)10
One event.

7.14.2.20 #define CH_STATE_WTANDEVT (tstate_t)11
Several events.

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

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

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

7.14.2.24 #define CH_STATE_FINAL (tstate_t)15
Thread terminated.

7.14.2.25 #define CH_STATE_NAMES

Value:


Thread states as array of strings.
Each element in an array initialized with this macro can be indexed using the numeric thread state values.
7.14.2.26  #define CH_FLAG_MODE_MASK (tmode_t)3U
Thread memory mode mask.

7.14.2.27  #define CH_FLAG_MODE_STATIC (tmode_t)0U
Static thread.

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

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

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

7.14.2.31  #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.

7.14.2.32  #define currp ch.rlist.current
Current thread pointer access macro.

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

7.14.2.33  #define __CH_STRINGIFY( a ) #a
Utility to make the parameter a quoted string.

7.14.3  Typedef Documentation

7.14.3.1  typedef struct ch_thread thread_t
Type of a thread structure.

7.14.3.2  typedef thread_t* thread_reference_t
Type of a thread reference.
typedef struct ch_threads_list threads_list_t
Type of a generic threads single link list, it works like a stack.

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

typedef struct ch_ready_list ready_list_t
Type of a ready list header.

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

typedef struct ch_virtual_timer virtual_timer_t
Type of a Virtual Timer structure.

typedef struct ch_virtual_timers_list virtual_timers_list_t
Type of virtual timers list header.

typedef struct ch_system_debug system_debug_t
Type of a system debug structure.

typedef struct ch_system ch_system_t
Type of system data structure.

7.14.4 Function Documentation

void _scheduler_init ( void )
Scheduler initialization.

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

Here is the call graph for this function:

```
 Scheduler initialization.

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

```

ChibiOS/RT
7.14.4.2  static void queue_prio_insert ( thread_t * tp, threads_queue_t * tqp )  [inline]

Inserts a thread into a priority ordered queue.

Note

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

Parameters

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

Function Class:

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

7.14.4.3  static void queue_insert ( thread_t * tp, threads_queue_t * tqp )  [inline]

Inserts a thread into a queue.

Parameters

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

Function Class:

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

7.14.4.4  static thread_t * queue_fifo_remove ( threads_queue_t * tqp )  [inline]

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

Note

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

Parameters

| in | tqp | the pointer to the threads list header               |

Returns

The removed thread pointer.

Function Class:

Not an API, this function is for internal use only.
7.14.5  static thread_t * queue_lifo_remove ( threads_queue_t * tqp ) [inline]

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

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

Parameters

| in | tqp | the pointer to the threads list header |

Returns
The removed thread pointer.

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

7.14.6  static thread_t * queue_dequeue ( thread_t * tp ) [inline]

Removes a thread from a queue and returns it.
The thread is removed from the queue regardless of its relative position and regardless the used insertion method.

Parameters

| in | tp | the pointer to the thread to be removed from the queue |

Returns
The removed thread pointer.

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

7.14.7  static void list_insert ( thread_t * tp, threads_list_t * tlp ) [inline]

Pushes a thread_t on top of a stack list.

Parameters

| in | tp | the pointer to the thread to be inserted in the list |
| in | tlp | the pointer to the threads list header |

Function Class:
Not an API, this function is for internal use only.
7.14.4.8  static thread_t * list_remove ( threads_list_t * tlp ) [inline]

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

Precondition

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

Parameters

| in  | tlp | the pointer to the threads list header |

Returns

The removed thread pointer.

Function Class:

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

7.14.4.9  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:

```
chSchReadyI    chDbgCheckClassI    chSysHalt
```

7.14.4.10  thread_t * chSchReadyAheadI ( thread_t * tp )

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

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

Precondition

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

Postcondition

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

Parameters

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

The thread pointer.

Function Class:

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

Here is the call graph for this function:

```
chSchReadyAheadI  
  ^           
  │           
  │           
  │           
  v           
chDbgCheckClassI  
  ^           
  │           
  │           
  │           
  v           
chSysHalt
```

7.14.4.11  void chSchGoSleepS ( tstate_t newstate )

Puts the current thread to sleep into the specified state.
The thread goes into a sleeping state. The possible **Thread States** are defined into *threads.h*.

Parameters

| in      | newstate | the new thread state |

Function Class:

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

Here is the call graph for this function:

```
chSchGoSleepS  
  ^           
  │           
  │           
  │           
  v           
chDbgCheckClassS
  ^           
  │           
  │           
  │           
  v           
queue_fifo_remove
  ^           
  │           
  │           
  │           
  v           
queue_dequeue
  ^           
  │           
  │           
  │           
  v           
chSchReadyI
  ^           
  │           
  │           
  │           
  v           
chDbgCheckClassS
  ^           
  │           
  │           
  │           
  v           
_stats_start_measure  
  │           
  │           
  │           
  v           
_crit_isr  
  ^           
  │           
  │           
  │           
  v           
_dbg_check_lock_from_isr  
  │           
  │           
  │           
  v           
_dbg_check_unlock_from_isr  
  │           
  │           
  │           
  v           
_stats_stop_measure  
  │           
  │           
  │           
  v           
_crit_isr  
  ^           
  │           
  │           
  │           
  v           
_dbg_check_lock_from_isr  
  │           
  │           
  │           
  v           
_dbg_check_unlock_from_isr  
  │           
  │           
  │           
  v           
_stats_stop_measure  
  │           
  │           
  │           
  v           
_crit_isr
```
7.14 Scheduler

7.14.4.12 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:
7.14.4.13 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></th>
<th>in ntp</th>
<th>the thread to be made ready</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>in msg</td>
<td>the wakeup message</td>
</tr>
</tbody>
</table>

Function Class:

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

Here is the call graph for this function:

7.14.4.14 void chSchRescheduleS ( void )

Performs a reschedule if a higher priority thread is runnable.

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

Function Class:

This is an S-Class API, this function can be invoked from within a system lock zone by threads only.
Here is the call graph for this function:

7.14.4.15 bool chSchlsPreemptionRequired ( void )

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

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

Return values

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

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

7.14.4.16 void chSchDoRescheduleBehind ( void )

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

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

Function Class:

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

Here is the call graph for this function:

```
chSchDoRescheduleBehind
\arrow{queue_fifo_remove}
\arrow{chSchReadyI}
\arrow{chDbgCheckClassI}
```

7.14.4.17 `void chSchDoRescheduleAhead ( void )`

Switches to the first thread on the runnable queue.
The current thread is positioned in the ready list ahead of all threads having the same priority.

Note

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

Function Class:

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

Here is the call graph for this function:

```
chSchDoRescheduleAhead
\arrow{queue_fifo_remove}
\arrow{chSchReadyAheadI}
\arrow{chDbgCheckClassI}
```

7.14.4.18 `void chSchDoReschedule ( void )`

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

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

Function Class:

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

Here is the call graph for this function:

```
chSchDoReschedule
queue_fifo_remove
chSchReadyI
chSchReadyAheadI
chDbgCheckClassI
```

7.14.4.19  static void list_init ( threads_list_t * tlp )  [inline],[static]

Threads list initialization.

Parameters

| in | tlp | pointer to the threads list object |

Function Class:

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

7.14.4.20  static bool list isempty ( threads_list_t * tlp )  [inline],[static]

Evaluates to true if the specified threads list is empty.

Parameters

| in | tlp | pointer to the threads list object |

Returns

The status of the list.

Function Class:

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

7.14.4.21  static bool list notempty ( threads_list_t * tlp )  [inline],[static]

Evaluates to true if the specified threads list is not empty.
Parameters

\[
\text{in} \quad tlp \quad \text{pointer to the threads list object}
\]

Returns

The status of the list.

Function Class:

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

7.14.4.22 static void queue_init (threads_queue_t *tqp) [inline],[static]

Threads queue initialization.

Parameters

\[
\text{in} \quad tqp \quad \text{pointer to the threads queue object}
\]

Function Class:

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

7.14.4.23 static bool queue_isempty (const threads_queue_t *tqp) [inline],[static]

Evaluates to true if the specified threads queue is empty.

Parameters

\[
\text{in} \quad tqp \quad \text{pointer to the threads queue object}
\]

Returns

The status of the queue.

Function Class:

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

Here is the call graph for this function:

```
queue_isempty -> list_insert
```
7.14.4.24 static bool queue_notempty ( const threads_queue_t * tqp ) [inline],[static]

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

Parameters

in  tqp  pointer to the threads queue object

Returns

The status of the queue.

Function Class:

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

Here is the call graph for this function:

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

Determines if the current thread must reschedule.

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

Returns

The priorities situation.

Return values

false  if rescheduling is not necessary.
true  if there is a ready thread at higher priority.
Function Class:

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

Here is the call graph for this function:

```
chSchIsRescRequiredI -> chDbgCheckClassS -> chSysHalt
```

### 7.14.4.26 static bool chSchCanYieldS ( void ) [inline],[static]

Determines if yielding is possible.

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

Returns

The priorities situation.

Return values

- `false` if yielding is not possible.
- `true` if there is a ready thread at equal or higher priority.

Function Class:

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

Here is the call graph for this function:

```
chSchCanYieldS -> chDbgCheckClassS -> chSysHalt
```

### 7.14.4.27 static void chSchDoYieldS ( void ) [inline],[static]

Yields the time slot.

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

Function Class:

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

Here is the call graph for this function:

```
chSchDoYieldS -> chSchDoRescheduleAhead -> queue_fifo_remove -> chSchReadyAheadI
```

7.14.4.28 static void chSchPreemption ( void ) [inline],[static]

Inline-able preemption code.

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

Function Class:

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

Here is the call graph for this function:

```
chSchPreemption -> chSchDoRescheduleBehind -> queue_fifo_remove -> chSchReadyI
```

7.14.5 Variable Documentation

7.14.5.1 ch_system_t ch

System data structures.
7.15 Time and Intervals

7.15.1 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_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 time before it wraps.

Fast time conversion utilities

- \#define TIME_S2I(secs) ((sysinterval_t)((time_conv_t)(secs) ∗ (time_conv_t)CH_CFG_ST_FREQUENCY))
  Seconds to time interval.
- \#define TIME_MS2I(msecs)
  Milliseconds to time interval.
- \#define TIME_US2I(usecs)
  Microseconds to time interval.
- \#define TIME_I2S(interval)
  Time interval to seconds.
- \#define TIME_I2MS(interval)
  Time interval to milliseconds.
- \#define TIME_I2US(interval)
  Time interval to microseconds.

Secure time conversion utilities

- static sysinterval_t chTimeS2I(time_secs_t secs)
  Seconds to time interval.
- static sysinterval_t chTimeMS2I(time_msecs_t msec)
  Milliseconds to time interval.
- static sysinterval_t chTimeUS2I(time_usecs_t usec)
  Microseconds to time interval.
- static time_secs_t chTimeI2S(sysinterval_t interval)
  Time interval to seconds.
- static time_msecs_t chTimeI2MS(sysinterval_t interval)
  Time interval to milliseconds.
- static time_usecs_t chTimeI2US(sysinterval_t interval)
  Time interval to microseconds.
- static systime_t chTimeAddX(systime_t systime, sysinterval_t interval)
  Adds an interval to a system time returning a system time.
- static sysinterval_t chTimeDiffX(systime_t start, systime_t end)
  Subtracts two system times returning an interval.
- static bool chTimesInRangeX(systime_t time, systime_t start, systime_t end)
  Checks if the specified time is within the specified time range.
TypeDefs

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

7.15.2 Macro Definition Documentation

7.15.2.1 #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.

7.15.2.2 #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.

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

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

7.15.2.5 #define TIME_S2I(secs) ((sysinterval_t)((time_conv_t)(secs) * (time_conv_t)CH_CFG_ST_FREQUENCY_CY))
Seconds to time interval.
Converts from seconds to system ticks number.

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

<table>
<thead>
<tr>
<th></th>
<th>secs</th>
<th>number of seconds</th>
</tr>
</thead>
</table>

Returns

The number of ticks.

Function Class:

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

7.15.2.6 #define TIME_MS2I( msecs )

Value:

\[
(sysinterval_t)\left\{((time\_conv\_t)(msecs) \times (time\_conv\_t)CH\_CFG\_ST\_FREQUENCY) + (time\_conv\_t)999) / (time\_conv\_t)1000\right\}
\]

Milliseconds to time interval.

Converts from milliseconds to system ticks number.

Note

The result is rounded upward to the next tick boundary.

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

Parameters

<table>
<thead>
<tr>
<th></th>
<th>msecs</th>
<th>number of milliseconds</th>
</tr>
</thead>
</table>

Returns

The number of ticks.

Function Class:

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

7.15.2.7 #define TIME_US2I( usecs )

Value:

\[
(sysinterval_t)\left\{((time\_conv\_t)(usecs) \times (time\_conv\_t)CH\_CFG\_ST\_FREQUENCY) + (time\_conv\_t)999999) / (time\_conv\_t)1000000\right\}
\]

Microseconds to time interval.

Converts from microseconds to system ticks number.
Note

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

\[ \text{in } \text{usecs} \ \text{number of microseconds} \]

Returns

The number of ticks.

Function Class:

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

7.15.2.8 \#define TIME_I2S( interval )

Value:

\[
\text{(timesecs_t)}\{\{(\text{time conv t})(\text{interval}) + \ (\text{time conv t})\text{CH CFG ST FREQUENCY} - \ (\text{time conv t})1\} / (\text{time conv t})\text{CH CFG ST FREQUENCY)}
\]

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

Note

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

Parameters

\[ \text{in } \text{interval} \ \text{interval in ticks} \]

Returns

The number of seconds.

Function Class:

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

7.15.2.9 \#define TIME_I2MS( interval )

Value:

\[
\text{(timemsecs_t)}\{\{(\text{time conv t})(\text{interval}) \times (\text{time conv t})1000) + \ (\text{time conv t})\text{CH CFG ST FREQUENCY} - \ (\text{time conv t})1\} / (\text{time conv t})\text{CH CFG ST FREQUENCY)}
\]

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

\texttt{in interval} \hspace{0.5cm} \text{interval in ticks}

Returns

The number of milliseconds.

Function Class:

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

7.15.2.10 \texttt{#define TIME_I2US( interval )}

\textbf{Value:}

\begin{verbatim}
(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}
\end{verbatim}

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

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

Parameters

\texttt{in interval} \hspace{0.5cm} \text{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.

7.15.3 Typedef Documentation

7.15.3.1 typedef uint64_t systime_t

Type of system time.
7.15.3.2 typedef uint64_t sysinterval_t
Type of time interval.

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

7.15.3.3 typedef uint32_t time_secs_t
Type of seconds.

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

7.15.3.4 typedef uint32_t time_msecs_t
Type of milliseconds.

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

7.15.3.5 typedef uint32_t time_usecs_t
Type of microseconds.

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

7.15.3.6 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.

7.15.4 Function Documentation

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

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

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

| in | secs | number of seconds |

>Returns

The number of ticks.

Function Class:

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

7.15.4.2 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.

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

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

Note

The result is rounded upward to the next tick boundary.

Parameters

| in | usec | number of microseconds |

>Returns

The number of ticks.

Function Class:

Special function, this function has special requirements see the notes.
7.15.4.4 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
\begin{verbatim}
in interval interval in ticks
\end{verbatim}

Returns
The number of seconds.

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

7.15.4.5 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
\begin{verbatim}
in interval interval in ticks
\end{verbatim}

Returns
The number of milliseconds.

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

7.15.4.6 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

\[
\text{in} \quad \text{interval} \quad \text{interval in ticks}
\]

Returns

The number of microseconds.

Function Class:

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

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

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

Parameters

\[
\text{in} \quad \text{systime} \quad \text{base system time}
\]
\[
\text{in} \quad \text{interval} \quad \text{interval to be added}
\]

Returns

The new system time.

Function Class:

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

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

Subtracts two system times returning an interval.

Parameters

\[
\text{in} \quad \text{start} \quad \text{first system time}
\]
\[
\text{in} \quad \text{end} \quad \text{second system time}
\]

Returns

The interval representing the time difference.

Function Class:

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

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

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

Note

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

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

Return values

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

Function Class:

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

7.16.1 Detailed Description

Time and Virtual Timers related APIs and services.

Functions

- **void _vt_init (void)**
  
  *Virtual Timers initialization.*

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

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

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

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

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

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

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

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

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

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

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

- **static void chVTDoTickI (void)**
  
  *Virtual timers ticker.*

7.16.2 Function Documentation

7.16.2.1 void _vt_init ( void )

*Virtual Timers initialization.*
Note

Internal use only.

Function Class:

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

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

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

Precondition

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

Note

The callback function is invoked from interrupt context.

Parameters

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

![Call graph diagram]

7.16.2.3 void chVTDoResetI ( virtual_timer_t * vtp )

Disables a Virtual Timer.

Precondition

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

Parameters

| in  | vtp | the virtual_timer_t structure pointer |

Function Class:

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

```
chVTDoReset
  ↓
  chTimeDiffX
  ↓
  chVTGetSystemTimeX
  ↓
  chTimeAddX
  ↓
  chDbgCheckClassI
  ↓
  chSysHalt
```

### 7.16.2.4 static void chVTObjetInit ( 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.

### 7.16.2.5 static systime_t chVTOGetSystemTimeX ( 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.
7.16 Virtual Timers

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

Current system time.

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

Note

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

Returns

The system time in ticks.

Function Class:

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

Here is the call graph for this function:

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

Returns the elapsed time since the specified start time.

Parameters

\[\text{in} \quad \text{start} \quad \text{start time}\]
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
```

7.16.2.8 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 true because the whole time range is specified.

Parameters

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

Return values

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

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

```
chVTIsSystemTimeWithin
    chTimeIsInRangeX
      chVTGetSystemTimeX
```

### 7.16.2.9 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 true because the whole time range is specified.

**Parameters**

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

**Return values**

| 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:

```
chVTIsSystemTimeWithin
    chTimeIsInRangeX
      chVTGetSystemTimeX
```
7.16.2.10  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:

```
chDbgCheckClassI  chSysHalt
chVTGetTimersStateI  chTimeDiffX
chVTGetSystemTimeX
```

7.16.2.11  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 chVTDSetI().
7.16 Virtual Timers

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
```

7.16.2.12 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 `chVToObjectInit()` or `chVDoSetI()`.

Parameters

| in  | vtp | the virtual_timer_t structure pointer |

ChibiOS/RT
Returns

true if the timer is armed.

Function Class:

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

Here is the call graph for this function:

- chVTIsArmed
- chSysLock
- chVTIsArmedI
- chSysUnlock
- _stats_start_measure
- _crit_thd
- _dbg_check_lock
- _dbg_check_unlock
- _stats_stop_measure
- _crit_thd

7.16.2.13 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:

### 7.16.2.14 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 `chVObjectInit()` or `chVDoSetI()`.

**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
chVTResetI
chSysUnlock
_stats_start_measure
_crit_thd
_dbg_check_lock
chVTIsArmedI
chVTDoResetI
_dbg_check_unlock
_stats_stop_measure
_crit_thd
```

7.16.2.15 static void chVTSetI ( virtual_timer_t ∗vtp, sysinterval_t delay, vtfunc_t vtfunc, void ∗par ) [inline],

[static]

Enables a virtual timer.

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

Precondition

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

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>- <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:

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

Enables a virtual timer.

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

**Precondition**

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

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>vtp</code></td>
<td>virtual_timer_t structure pointer</td>
</tr>
<tr>
<td><code>delay</code></td>
<td>the number of ticks before the operation timeouts, the special values are handled as follow:</td>
</tr>
<tr>
<td></td>
<td>• <code>TIME_INFINITE</code> 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><code>vtfunc</code></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>a parameter that will be passed to the callback function</td>
</tr>
</tbody>
</table>

**Function Class:**

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

7.16.2.17 static void chVTDoTickI ( void ) [inline],[static]

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:
7.17 Threads

7.17.1 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) {(thread_t *)&name, (thread_t *)&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), P ← ORT_STACK_ALIGN)`
  Calculates the total Working Area size.
- `#define THD_WORKING_AREA(s, n) PORT_WORKING_AREA(s, n)`
  Static working area allocation.
- `#define THD_WORKING_AREA_BASE(s) ((stkalign_t *)(s))`
  Base of a working area casted to the correct type.
- `#define THD_WORKING_AREA_END(s)`
  End of a working area casted to the correct type.

Threads abstraction macros

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

- #define chThdSleepSeconds(sec) chThdSleep(TIME_S2I(sec))
  Delays the invoking thread for the specified number of seconds.
- #define chThdSleepMilliseconds(msec) chThdSleep(TIME_MS2I(msec))
  Delays the invoking thread for the specified number of milliseconds.
- #define chThdSleepMicroseconds(usec) chThdSleep(TIME_US2I(usec))
  Delays the invoking thread for the specified number of microseconds.

Typedefs

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

Data Structures

- struct thread_descriptor_t
  Type of a thread descriptor.

Functions

- thread_t * _thread_init (thread_t *tp, const char *name, tprio_t prio)
  Initializes a thread structure.
- void _thread_memfill (uint8_t *startp, uint8_t *endp, uint8_t v)
  Memory fill utility.
- thread_t * chThdCreateSuspendedI (const thread_descriptor_t *tdp)
  Creates a new thread into a static memory area.
- thread_t * chThdCreateSuspended (const thread_descriptor_t *tdp)
  Creates a new thread into a static memory area.
- thread_t * chThdCreateI (const thread_descriptor_t *tdp)
  Creates a new thread into a static memory area.
- thread_t * chThdCreate (const thread_descriptor_t *tdp)
  Creates a new thread into a static memory area.
- 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 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.
### 7.17.2 Macro Definition Documentation

#### 7.17.2.1 
```
#define _THREADS_QUEUE_DATA( name ) ((thread_t *)&name, (thread_t *)&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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>name</strong></td>
<td>the name of the threads queue variable</td>
</tr>
</tbody>
</table>

#### 7.17.2.2 
```
#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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>name</strong></td>
<td>the name of the threads queue variable</td>
</tr>
</tbody>
</table>

#### 7.17.2.3 
```
#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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>n</strong></td>
<td>the stack size to be assigned to the thread</td>
</tr>
</tbody>
</table>

**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.

#### 7.17.2.4 
```
#define THD_WORKING_AREA( s, n ) PORT_WORKING_AREA(s, n)
```

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

**Parameters**

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

**Function Class:**

Normal API, this function can be invoked by regular system threads but not from within a lock zone.
7.17.2.5  
#define THD_WORKING_AREA_BASE(s) ((stkalign_t*)(s))

Base of a working area casted to the correct type.

Parameters

| in  | s   | name of the working area |

7.17.2.6  
#define THD_WORKING_AREA_END(s)

Value:

\( (\text{THD\_WORKING\_AREA\_BASE}(\text{s}) + \text{(sizeof (s)) / sizeof (stkalign_t)}) \)

End of a working area casted to the correct type.

Parameters

| in  | s   | name of the working area |

7.17.2.7  
#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.

7.17.2.8  
#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.
7.17.9 \#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.

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

Delays the invoking thread for the specified number of microseconds.

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

Parameters

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

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

7.17.3 Typedef Documentation

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

7.17.4 Function Documentation

7.17.4.1 thread_t * _thread_init ( thread_t * tp, const char * name, tprio_t prio )
Initializes a thread structure.

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

| in | tp       | pointer to the thread |
|    | name     | thread name           |
|    | 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:

```
<table>
<thead>
<tr>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>_thread_init</td>
</tr>
<tr>
<td>list_init</td>
</tr>
<tr>
<td>queue_init</td>
</tr>
<tr>
<td>chTMObjectInit</td>
</tr>
</tbody>
</table>
```

7.17.4.2  void _thread_memfill ( uint8_t * startp, uint8_t * endp, uint8_t v )

Memory fill utility.

Parameters

| in | startp | first address to fill |
|    | endp   | last address to fill +1 |
|    | v      | filler value |

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

7.17.4.3  thread_t * chThdCreateSuspended ( const thread_descriptor_t * tdp )

Creates a new thread into a static memory area.
The new thread is initialized but not inserted in the ready list, the initial state is CH_STATE_WTSTART.
Postcondition

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

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

Note

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

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

Parameters

\begin{itemize}
  \item \textbf{out} tdp pointer to the thread descriptor
\end{itemize}

Returns

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

Function Class:

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

Here is the call graph for this function:

\begin{center}
\begin{tikzpicture}
  \node (root) at (0,0) {\texttt{chThdCreateSuspendedI}};
  \node (chk) [above of=root] {\texttt{chDbgCheckClassI}};
  \node (halt) [right of=root] {\texttt{chSysHalt}};
  \node (thread) [below of=root] {\_thread\_init};
  \node (list) [right of=thread] {\texttt{list\_init}};
  \node (queue) [right of=list] {\texttt{queue\_init}};
  \node (object) [below of=queue] {\texttt{chTMObjectInit}};

  \draw[->] (root) -- (chk);
  \draw[->] (root) -- (halt);
  \draw[->] (chk) -- (thread);
  \draw[->] (thread) -- (list);
  \draw[->] (list) -- (queue);
  \draw[->] (queue) -- (object);
\end{tikzpicture}
\end{center}

\subsection{7.17.4.4 \texttt{thread\_t * chThdCreateSuspended ( const thread\_descriptor\_t * tdp )}}

Creates a new thread into a static memory area.

The new thread is initialized but not inserted in the ready list, the initial state is \texttt{CH\_STATE\_WTSTART}. 

---

ChibiOS/RT
Postcondition

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

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

Note

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

Parameters

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

Returns

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

Function Class:

Normal API, 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{verbatim}
chThdCreateSuspended
chRegFindThreadByWorkingArea
chSysLock
chSysUnlock
_thread_memfill
chThdCreateSuspendedI
chRegFirstThread
chThdGetWorkingAreaX
chRegNextThread
chDbgCheckClassI
_thread_init

7.17.4.5 \ 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 \texttt{chThdRelease()} or \texttt{chthdWait()} in order to release the reference. The thread persists in the registry until its reference counter reaches zero.

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

A thread can terminate by calling `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:

```
chThdCreateI
  |         |
  |         | chSchReadyI
  |         | chDbgCheckClassI
  v         v
chThdCreateSuspendedI
  |            |
  |            | _thread_init
```

7.17.4.6 `thread_t * chThdCreate ( 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.

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.

Here is the call graph for this function:

![Call Graph](image)

### 7.17.4.7 thread_t * chThdCreateStatic ( void * wsp, size_t size, tprio_t prio, tfunc_t pf, void * arg )

Creates a new thread into a static memory area.

**Postcondition**

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

**Note**

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

**Parameters**

<table>
<thead>
<tr>
<th><strong>Parameter</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>wsp</code></td>
<td>pointer to a working area dedicated to the thread stack</td>
</tr>
<tr>
<td><code>size</code></td>
<td>size of the working area</td>
</tr>
<tr>
<td><code>prio</code></td>
<td>the priority level for the new thread</td>
</tr>
<tr>
<td><code>pf</code></td>
<td>the thread function</td>
</tr>
<tr>
<td><code>arg</code></td>
<td>an argument passed to the thread function. It can be <code>NULL</code>.</td>
</tr>
</tbody>
</table>
7.17 Threads

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:

7.17.4.8 thread_t * chThdStart ( thread_t * tp )

Resumes a thread created with chThdCreateI().

Parameters

| in     | tp   | pointer to the thread |
Returns

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

Function Class:

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

Here is the call graph for this function:

```
chThdStart
  \rightarrow chSysLock
    \rightarrow _dbg_check_lock
      \rightarrow chDbgCheckClassS
        \rightarrow chSchWakeupS
          \rightarrow chSchReady!
            \rightarrow chSchReadyAhead!
              \rightarrow chSysUnlock
                \rightarrow _dbg_check_unlock
                  \rightarrow _stats_stop_measure
                    \rightarrow _crit_thd
                      \rightarrow _stats_start_measure
                        \rightarrow _crit_thd
```

7.17.4.9  `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.
7.17 Threads

Here is the call graph for this function:

```
chThdAddRef
  ↘
  └── chSysLock
        ↘
        └── _stats_start_measure
            └── _crit_thd

chSysUnlock
  ↘
  └── _dbg_check_lock

chThdAddRef
  ↘
  └── _dbg_check_unlock

chThdAddRef
  ↘
  └── _stats_stop_measure
        └── _crit_thd
```

7.17.4.10 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

<table>
<thead>
<tr>
<th>in</th>
<th>tp</th>
<th>pointer to the thread</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 Diagram]

7.17.4.11 void chThdExit ( msg_t msg )

Terminates the current thread.

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

Postcondition

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

Parameters

| in  | msg | thread exit code |

Function Class:

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

Here is the call graph for this function:

![Call Graph Diagram]

7.17.4.12 void chThdExitS ( msg_t msg )

Terminates the current thread.

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

Postcondition

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

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

Parameters

| in   | msg | thread exit code |

Function Class:

This is an S-Class API, this function can be invoked from within a system lock zone by threads only.
7.17.4.13  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:

7.17.4.14  

`tprio_t chThdSetPriority ( tprio_t newprio )`

Changes the running thread priority level then reschedules if necessary.

Note

The function returns the real thread priority regardless of the current priority that could be higher than the real priority because the priority inheritance mechanism.

Parameters

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

The old priority level.

Function Class:

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

Here is the call graph for this function:

```
7.17.4.15 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:

7.17.4.16 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:

7.17.4.17 void chThdSleepUntil ( systime_t time )

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

Note

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

See also

chThdSleepUntilWindowed()

Parameters

| in  | time | absolute system time |

Function Class:

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

```
chThdSleepUntil
  |     |
  |     |  _stats_start_measure
  |     |  _crit_thd
  |     |
  |     |  chSysLock
  |     |  _dbg_check_lock
  |     |
  |     |  chTimeDiffX
  |     |  _dbg_check_unlock
  |     |
  |     |  chVTGetSystemTimeX
  |     |
  |     |  chThdSleepS
  |     |  chSchGoSleepTimeoutS
  |     |
  |     |  chSysUnlock
  |     |  _stats_stop_measure
  |     |  _crit_thd
```

7.17.4.18  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 time else the call is assumed to have been called outside the allowed time interval, in this case no sleep is performed.

See also

chThdSleepUntil()  

Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>prev</td>
</tr>
<tr>
<td>in</td>
<td>next</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:

7.17.4.19  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:

```
7.17.4.20 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:

![Call Graph Diagram]

7.17.4.21 `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>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>trp</td>
<td>a pointer to a thread reference object</td>
</tr>
<tr>
<td>timeout</td>
<td>the timeout in system ticks, the special values are handled as follow:</td>
</tr>
<tr>
<td></td>
<td>- <code>TIME_INFINITE</code> the thread enters an infinite sleep state.</td>
</tr>
<tr>
<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 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)

### 7.17.4.22 void chThdResumeI (thread_reference_t *trp, msg_t msg )

Wakes up a thread waiting on a thread reference object.

**Note**

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

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>trp</td>
<td>a pointer to a thread reference object</td>
</tr>
<tr>
<td>in</td>
<td>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](image)

### 7.17.4.23 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>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>trp</td>
<td>thread_reference_t*</td>
<td>a pointer to a thread reference object</td>
</tr>
<tr>
<td>msg</td>
<td>msg_t</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:

```
chThdResumeS → chSchWakeupS → chSchReadyI → chSchReadyAheadI → chDbgCheckClassS
```

7.17.4.24  void chThdResume (thread_reference_t *trp, msg_t msg)

Wakes up a thread waiting on a thread reference object.

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

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>trp</td>
<td>thread_reference_t*</td>
<td>a pointer to a thread reference object</td>
</tr>
<tr>
<td>msg</td>
<td>msg_t</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:

7.17.4.25 msg_t chThdEnqueueTimeoutS ( threads_queue_t * tqp, sysinterval_t timeout )

Enqueues the caller thread on a threads queue object.
The caller thread is enqueued and put to sleep until it is dequeued or the specified timeouts expires.

Parameters

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

Returns

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

Return values

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

Function Class:

This is an S-Class API, this function can be invoked from within a system lock zone by threads only.
Here is the call graph for this function:

```
void chThdDequeueNextI (threads_queue_t *tp, msg_t msg)
```

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

**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:

```
void chThdDequeueAllI (threads_queue_t *tp, msg_t msg)
```

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

Parameters

| in  | msg | pointer to the threads queue object
| in  | msg | the message code

Function Class:

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

Here is the call graph for this function:

![Call Graph](image)

7.17.4.28 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.

7.17.4.29 static tprio_t chThdGetPriorityX ( void ) [inline],[static]

Returns the current thread priority.

Note

Can be invoked in any context.
Returns

The current thread priority.

Function Class:

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

Here is the call graph for this function:

```
chThdGetPriorityX chThdGetSelfX
```

### 7.17.4.30 static systime_t chThdGetTicksX ( thread_t * tp ) [inline],[static]

Returns the number of ticks consumed by the specified thread.

Note

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

Parameters

- **in tp** pointer to the thread

Returns

The number of consumed system ticks.

Function Class:

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

### 7.17.4.31 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.
7.17.4.32 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

| true | thread terminated. |
| false | thread not terminated. |

Function Class:

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

7.17.4.33 static bool chThdShouldTerminateX ( void ) [inline],[static]

Verifies if the current thread has a termination request pending.

Return values

| true | termination request pending. |
| false | termination request not pending. |

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]

7.17.4.34 static thread_t* chThdStartI ( thread_t *tp ) [inline],[static]

Resumes a thread created with chThdCreateI().

Parameters

| in  | tp | pointer to the thread |

ChibiOS/RT
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 chDbgCheckClassI
```

7.17.4.35 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>

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:

```
chThdStartI chSchGoSleepTimeoutS chDbgCheckClassS
```

ChibiOS/RT
7.17.4.36 static void chThdQueueObjectInit ( threads_queue_t * tqp ) [inline],[static]

Initializes a threads queue object.

Parameters

| out | tqp | pointer to the threads queue object |

Function Class:

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

Here is the call graph for this function:

chThdQueueObjectInit -> queue_init

7.17.4.37 static bool chThdQueueIsEmptyI ( threads_queue_t * tqp ) [inline],[static]

Evaluates to true if the specified queue is empty.

Parameters

| out | tqp | pointer to the threads queue object |

Returns

The queue status.

Return values

| false | if the queue is not empty. |
| true  | if the queue is empty. |

Function Class:

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

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

```

7.17.4.38 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:
7.18 Time Measurement

7.18.1 Detailed Description

Time Measurement APIs and services.

Data Structures

- `struct tm_calibration_t`
  
  Type of a time measurement calibration data.

- `struct time_measurement_t`
  
  Type of a Time Measurement object.

Functions

- `void _tm_init (void)`
  
  Initializes the time measurement unit.

- `void chTMObjectInit (time_measurement_t *tmp)`
  
  Initializes a TimeMeasurement object.

- `NOINLINE void chTMStartMeasurementX (time_measurement_t *tmp)`
  
  Starts a measurement.

- `NOINLINE void chTMStopMeasurementX (time_measurement_t *tmp)`
  
  Stops a measurement.

- `NOINLINE void chTMChainMeasurementToX (time_measurement_t *tmp1, time_measurement_t *tmp2)`
  
  Stops a measurement and chains to the next one using the same time stamp.

7.18.2 Function Documentation

7.18.2.1 `void _tm_init ( void )`

Initializes the time measurement unit.

Function Class:

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

Here is the call graph for this function:

```
_chm_init
    _tm_init
        chTMObjectInit
            chTMStartMeasurementX
                chTMStopMeasurementX
```

ChibiOS/RT
7.18 Time Measurement

7.18.2.2 void chTMObjectInit ( time_measurement_t * tmp )

Initializes a TimeMeasurement object.

Parameters

| out | tmp | pointer to a TimeMeasurement structure |

Function Class:

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

7.18.2.3 NOINLINE void chTMStartMeasurementX ( time_measurement_t * tmp )

Starts a measurement.

Precondition

The time_measurement_t structure must be initialized.

Parameters

| in,out | tmp | pointer to a TimeMeasurement structure |

Function Class:

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

7.18.2.4 NOINLINE void chTMStopMeasurementX ( time_measurement_t * tmp )

Stops a measurement.

Precondition

The time_measurement_t structure must be initialized.

Parameters

| in,out | tmp | pointer to a time_measurement_t structure |

Function Class:

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

7.18.2.5 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

| in,out | tmp1 | pointer to the time_measurement_t structure to be stopped |
| in,out | tmp2 | pointer to the time_measurement_t structure to be started |
Function Class:

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

7.19.1 Detailed Description

Synchronization services.

Modules

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

7.20.1 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) (_THREADS_QUEUE_DATA(name.queue), n)`
  Data part of a static semaphore initializer.
- `#define SEMAPHORE_DECL(name, n) semaphore_t name = _SEMAPHORE_DATA(name, n)`
  Static semaphore initializer.

Typedefs

- `typedef struct ch_semaphore semaphore_t`
  Semaphore structure.

Data Structures

- `struct ch_semaphore`
  Semaphore structure.
7.20 Counting Semaphores

Functions

- void chSemObjectInit (semaphore_t *sp, cnt_t n)
  - Initializes a semaphore with the specified counter value.
- void chSemReset (semaphore_t *sp, cnt_t n)
  - Performs a reset operation on the semaphore.
- void chSemResetI (semaphore_t *sp, cnt_t n)
  - 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 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.

7.20.2 Macro Definition Documentation

7.20.2.1 #define _SEMAPHORE_DATA(name, n) {
  _THREADS_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</th>
<th>name</th>
<th>the name of the semaphore variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>n</td>
<td>the counter initial value, this value must be non-negative</td>
</tr>
</tbody>
</table>

7.20.2.2 #define SEMAPHORE_DECL(name, n) semaphore_t name = _SEMAPHORE_DATA(name, n)

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

ChibiOS/RT
Module Documentation

Parameters

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

7.20.3 Typedef Documentation

7.20.3.1 typedef struct ch_semaphore semaphore_t

Semaphore structure.

7.20.4 Function Documentation

7.20.4.1 void chSemObjectInit ( semaphore_t * sp, cnt_t n )

Initializes a semaphore with the specified counter value.

Parameters

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

Function Class:

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

Here is the call graph for this function:

chSemObjectInit \(\rightarrow\) queue_init

7.20.4.2 void chSemReset ( semaphore_t * sp, cnt_t n )

Performs a reset operation on the semaphore.

Postcondition

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

Note

The released threads can recognize they were waked up by a reset rather than a signal because the chSemWait () will return MSG_RESET instead of MSG_OK.
7.20 Counting Semaphores

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:

Normal API, 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)

7.20.4.3 void chSemResetI (semaphore_t *sp, cnt_t n)

Performs a reset operation on the semaphore.

Postcondition

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

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

Note

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

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:

```
7.20.4.4 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().
Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.
```
7.20.4.5 `msg_t chSemWaitS ( semaphore_t * sp )`

Performs a wait operation on a semaphore.

Parameters

* `sp` pointer to a `semaphore_t` structure

Returns

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

Return values

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

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:

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

Performs a wait operation on a semaphore with timeout specification.

**Parameters**

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

**Returns**

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

**Return values**

- **MSG_OK** if the 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.
7.20 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:

```
chSemWaitTimeout
chSysLock
chSemWaitTimeoutS
chSysUnlock
_stats_start_measure
_crit_thd
_dbg_check_lock
chDbgCheckClassS
queue_isempty
queue_notempty
chSchGoSleepTimeoutS
_dbg_check_unlock
_stats_stop_measure
_crit_thd
```

7.20.4.7 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.

ChibiOS/RT
Here is the call graph for this function:

```
7.20.4.8 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.
7.20.4.9  void chSemSignal ( semaphore_t * sp )

Performs a signal operation on a semaphore.

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

Parameters

\[ \text{in} \quad \text{sp} \quad \text{pointer to a semaphore_t structure} \]
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:

```
chDbgCheckClassI
queue_isempty
queue_notempty
queue_fifo_remove
chSchReadyI
chSysHalt
list_insert
queue_prio_insert
queue_insert
queue_lifo_remove
queue_dequeue
chSchIsRescRequiredI
```

7.20.4.10 void chSemAddCounterI ( semaphore_t * sp, cnt_t n )

Adds the specified value to the semaphore counter.

Postcondition

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

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>sp</th>
<th>pointer to a semaphore_t structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>n</td>
<td>value to be added to the semaphore counter. The value must be positive.</td>
</tr>
</tbody>
</table>

Function Class:

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

Here is the call graph for this function:

```
chSemAddCounterI
chDbgCheckClassI
queue_isempty
queue_notempty
queue_fifo_remove
chSchReadyI
chSysHalt
list_insert
queue_prio_insert
queue_insert
queue_lifo_remove
queue_dequeue
chSchIsRescRequiredI
```

7.20.4.11 msg_t chSemSignalWait ( semaphore_t *sps, semaphore_t *spw )

Performs atomic signal and wait operations on two semaphores.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in sps</td>
<td>pointer to a semaphore_t structure to be signaled</td>
</tr>
<tr>
<td>in spw</td>
<td>pointer to a semaphore_t structure to wait on</td>
</tr>
</tbody>
</table>

Returns

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

Return values

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSG_OK</td>
<td>if the thread has not stopped on the semaphore or the semaphore has been signaled.</td>
</tr>
<tr>
<td>MSG_RESET</td>
<td>if the semaphore has been reset using chSemReset().</td>
</tr>
</tbody>
</table>

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.
7.20.4.12 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:

```
chSemFastWaitI  chDbgCheckClassl  chSysHalt
```

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

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

Parameters

| in | sp | pointer to a semaphore_t structure |

Function Class:

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

Here is the call graph for this function:

```
chSemFastSignalI  chDbgCheckClassl  chSysHalt
```

7.20.4.14  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 |

ChibiOS/RT
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:
7.21 Mutexes

7.21.1 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) (_THREADS_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 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.

7.21.2  Macro Definition Documentation

7.21.2.1  #define _MUTEX_DATA( name ) {

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

<table>
<thead>
<tr>
<th>Type</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>name</td>
<td>the name of the mutex variable</td>
</tr>
</tbody>
</table>
7.21 Mutexes

7.21.2.2 #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 |

7.21.3 Typedef Documentation

7.21.3.1 typedef struct ch_mutex mutex_t

Type of a mutex structure.

7.21.4 Function Documentation

7.21.4.1 void chMtxObjectInit ( mutex_t * mp )

Initializes s mutex_t structure.

Parameters

| out | mp | pointer to a mutex_t structure |

Function Class:

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

Here is the call graph for this function:

```
chMtxObjectInit  queue_init
```

7.21.4.2 void chMtxLock ( mutex_t * mp )

Locks the specified mutex.

Postcondition

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

Parameters

| in | mp | pointer to the mutex_t structure |
Function Class:

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

Here is the call graph for this function:

```
    chMtxLock
    chSysLock
    _stats_start_measure
    _crit_thd
    _dbg_check_lock
    chDbgCheckClassS
    queue_prio_insert
    queue_dequeue
    chSchReadyI
    _dbg_check_unlock
    _stats_stop_measure
    _crit_thd
    chMtxLock
    chMtxLock
    _stats_start_measure
    _crit_thd
    queue_prio_insert
    queue_dequeue
    chSchGoSleepS
    chSchReadyI
    _stats_stop_measure
    _crit_thd
    _dbg_check_unlock
    _crit_thd
    chSysUnlock
```

7.21.4.3  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.
7.21.4  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

\textbf{in} \hspace{1em} mp \hspace{1em} \textbf{pointer to the mutex_t structure}

Returns

The operation status.

Return values

\begin{tabular}{|l|l|}
\hline
\textbf{true} & if the mutex has been successfully acquired \\
\textbf{false} & if the lock attempt failed. \\
\hline
\end{tabular}

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.
7.21.4.5 \textbf{bool chMtxTryLockS ( \texttt{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.

\textbf{Postcondition}

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

\textbf{Note}

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

\textbf{Parameters}

\begin{itemize}
  \item \textbf{in mp} \hspace{1em} pointer to the \texttt{mutex_t} structure
\end{itemize}

\textbf{Returns}

The operation status.

\textbf{Return values}

\begin{itemize}
  \item \textbf{true} if the mutex has been successfully acquired
  \item \textbf{false} if the lock attempt failed.
\end{itemize}
Function Class:

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

Here is the call graph for this function:

```
chMtxTryLockS -> chDbgCheckClassS -> chSysHalt
```

### 7.21.4.6 void chMtxUnlock ( mutex_t *mp )

Unlocks the specified mutex.

**Note**

Mutexes must be unlocked in reverse lock order. Violating this rules will result in a panic if assertions are enabled.

**Precondition**

The invoking thread **must** have at least one owned mutex.

**Postcondition**

The mutex is unlocked and removed from the per-thread stack of owned mutexes.

**Parameters**

- **in mp**  
  pointer to the `mutex_t` structure

Function Class:

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

```
7.21.4.7 void chMtxUnlockS ( mutex_t *mp )
```

Unlocks the specified mutex.

**Note**

Mutexes must be unlocked in reverse lock order. Violating this rules will result in a panic if assertions are enabled.

**Precondition**

The invoking thread **must** have at least one owned mutex.

**Postcondition**

The mutex is unlocked and removed from the per-thread stack of owned mutexes. This function does not reschedule so a call to a rescheduling function must be performed before unlocking the kernel.

**Parameters**

- **in mp** pointer to the mutex_t structure

**Function Class:**

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

![Call Graph]

7.21.4.8  void chMtxUnlockAllS ( void )

Unlocks all mutexes owned by the invoking thread.

**Postcondition**

The stack of owned mutexes is emptied and all the found mutexes are unlocked. This function does not reschedule so a call to a rescheduling function must be performed before unlocking the kernel.

**Note**

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

**Function Class:**

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

Here is the call graph for this function:

![Call Graph]

7.21.4.9  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:

```
7.21.4.10 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](image)

7.21.4.11 static thread_t* chMtxGetOwnerI (mutex_t *mp) [inline],[static]

Returns the mutex owner thread.

Parameters

<table>
<thead>
<tr>
<th>out</th>
<th>mp</th>
<th>pointer to a mutex_t structure</th>
</tr>
</thead>
</table>

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
```

### 7.21.4.12 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
```
7.22 Condition Variables

7.22.1 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) (_THREADS_QUEUE_DATA(name.queue))
  Data part of a static condition variable initializer.
- #define CONDVAR_DECL(name) condition_variable_t name = _CONDVAR_DATA(name)
  Static condition variable initializer.

Typedefs

- typedef struct condition_variable condition_variable_t
  condition_variable_t structure.

Data Structures

- struct condition_variable
  condition_variable_t structure.

Functions

- void chCondObjectInit (condition_variable_t *cp)
  Initializes a condition_variable_t structure.
- void chCondSignal (condition_variable_t *cp)
  Signals one thread that is waiting on the condition variable.
- void chCondSignalI (condition_variable_t *cp)
  Signals one thread that is waiting on the condition variable.
- void chCondBroadcast (condition_variable_t *cp)
  Signals all threads that are waiting on the condition variable.
- void chCondBroadcastI (condition_variable_t *cp)
  Signals all threads that are waiting on the condition variable.
- msg_t chCondWait (condition_variable_t *cp)
  Waits on the condition variable releasing the mutex lock.
- msg_t chCondWaitS (condition_variable_t *cp)
  Waits on the condition variable releasing the mutex lock.
- msg_t chCondWaitTimeout (condition_variable_t *cp, sysinterval_t timeout)
Waits on the condition variable releasing the mutex lock.

- `msg_t chCondWaitTimeoutS (condition_variable_t *cp, sysinterval_t timeout)`
  
  Waits on the condition variable releasing the mutex lock.

### 7.22.2 Macro Definition Documentation

#### 7.22.2.1 `#define _CONDVAR_DATA( name ) {__THREADS_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**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>name</td>
<td>the name of the condition variable</td>
</tr>
</tbody>
</table>

#### 7.22.2.2 `#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**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>name</td>
<td>the name of the condition variable</td>
</tr>
</tbody>
</table>

### 7.22.3 Typedef Documentation

#### 7.22.3.1 `typedef struct condition_variable condition_variable_t`

`condition_variable_t` structure.

### 7.22.4 Function Documentation

#### 7.22.4.1 `void chCondObjectInit(condition_variable_t * cp)`

Initializes a `condition_variable_t` structure.

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>cp</td>
<td>pointer to a <code>condition_variable_t</code> structure</td>
</tr>
</tbody>
</table>

**Function Class:**

- **Initializer**, this function just initializes an object and can be invoked before the kernel is initialized.
7.22 Condition Variables

Here is the call graph for this function:

![Call Graph Diagram]

7.22.4.2 `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:

![Call Graph for chCondSignal]()

### 7.22.4.3 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.
7.22.4.4 void chCondBroadcast ( condition_variable_t * cp )

Signals all threads that are waiting on the condition variable.

Parameters

\textbf{in} \hspace{0.5cm} cp \hspace{0.5cm} \text{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:

### 7.22.4.5 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.
7.22.4.6 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

\[ \text{in } \text{cp} \] pointer to the condition_variable_t structure

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

Return values

<table>
<thead>
<tr>
<th>MSG_OK</th>
<th>if the condition variable has been signaled using chCondSignal().</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSG_RESET</td>
<td>if the condition variable has been signaled using chCondBroadcast().</td>
</tr>
</tbody>
</table>

Function Class:
Normal API, 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]

### 7.22.4.7 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.
7.22 Condition Variables

Here is the call graph for this function:

```
7.22.4.8 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>in</th>
<th>cp</th>
<th>pointer to the condition_variable_t structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>timeout</td>
<td>the number of ticks before the operation timeouts, the special values are handled as follow:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• TIME_INFINITE no timeout.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• TIME_IMMEDIATE this value is not allowed.</td>
</tr>
</tbody>
</table>

Returns

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

<table>
<thead>
<tr>
<th>MSG_OK</th>
<th>if the condition variable has been signaled using chCondSignal().</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSG_RESET</td>
<td>if the condition variable has been signaled using chCondBroadcast().</td>
</tr>
<tr>
<td>MSG_TIMEOUT</td>
<td>if the condition variable has not been signaled within the specified timeout.</td>
</tr>
</tbody>
</table>

Function Class:

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

Here is the call graph for this function:

```
7.22.4.9 msg_t chCondWaitTimeoutS ( condition_variable_t * cp, sysinterval_t timeout )
```

Waits on the condition variable releasing the mutex lock.

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

Precondition

The invoking thread must have at least one owned mutex.

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

Postcondition

Exiting the function because a timeout does not re-acquire the mutex, the mutex ownership is lost.
Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>cp</th>
<th>pointer to the condition_variable_t structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>timeout</td>
<td>the number of ticks before the operation timeouts, the special values are handled as follow:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• TIME_INFINITE no timeout.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• TIME_IMMEDIATE this value is not allowed.</td>
</tr>
</tbody>
</table>

Returns

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

Return values

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

Function Class:

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

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

7.23.1  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 become pending.
- **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 (∗)(eventid_t id))(&name)}
  Data part of a static event source initializer.
- #define EVENTSOURCE_DECL(name) event_source_t name = _EVENTSOURCE_DATA(name)
  Static event source initializer.

Typedefs

- typedef struct event_source event_source_t
  Event Source structure.
- typedef void (∗ evhandler_t) (eventid_t id)
  Event Handler callback function.
Data Structures

- struct event_listener
  Event Listener structure.
- struct event_source
  Event Source structure.

Functions

- void chEvtRegisterMaskWithFlags (event_source_t *esp, event_listener_t *elp, eventmask_t events, eventflags_t wflags)
  Registers an Event Listener on an Event Source.
- void chEvtUnregister (event_source_t *esp, event_listener_t *elp)
  Unregisters an Event Listener from its Event Source.
- eventmask_t chEvtGetAndClearEventsI (eventmask_t events)
  Clears the pending events specified in the events mask.
- eventmask_t chEvtGetAndClearEvents (eventmask_t events)
  Clears the pending events specified in the events mask.
- eventmask_t chEvtAddEvents (eventmask_t events)
  Adds (OR) a set of events to the current thread, this is much faster than using chEvtBroadcast() or chEvtSignal().
- void chEvtBroadcastFlagsI (event_source_t *esp, eventflags_t flags)
  Signals all the Event Listeners registered on the specified Event Source.
- eventflags_t chEvtGetAndClearFlags (event_listener_t *elp)
  Returns the flags associated to an event_listener_t.
- void chEvtSignal (thread_t *tp, eventmask_t events)
  Adds a set of event flags directly to the specified thread_t.
- void chEvtSignalI (thread_t *tp, eventmask_t events)
  Adds a set of event flags directly to the specified thread_t.
- void chEvtBroadcastFlags (event_source_t *esp, eventflags_t flags)
  Signals all the Event Listeners registered on the specified Event Source.
- eventflags_t chEvtGetAndClearFlagsI (event_listener_t *elp)
  Returns the unmasked flags associated to an event_listener_t.
- void chEvtDispatch (const evhandler_t *handlers, eventmask_t events)
  Invokes the event handlers associated to an event flags mask.
- eventmask_t chEvtWaitOne (eventmask_t events)
  Waits for exactly one of the specified events.
- eventmask_t chEvtWaitAny (eventmask_t events)
  Waits for any of the specified events.
- eventmask_t chEvtWaitAll (eventmask_t events)
  Waits for all the specified events.
- eventmask_t chEvtWaitOneTimeout (eventmask_t events, sysinterval_t timeout)
  Waits for exactly one of the specified events.
- eventmask_t chEvtWaitAnyTimeout (eventmask_t events, sysinterval_t timeout)
  Waits for any of the specified events.
- eventmask_t chEvtWaitAllTimeout (eventmask_t events, sysinterval_t timeout)
  Waits for all the specified events.
- 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 chEvt←
  Signal().
• static eventmask_t chEvtGetEventsX (void)
  Returns the events mask.

7.23.2  Macro Definition Documentation

7.23.2.1  #define ALL_EVENTS ((eventmask_t)-1)

All events allowed mask.

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

Returns an event mask from an event identifier.

7.23.2.3  #define _EVENTSOURCE_DATA( name ) { (event_listener_t *)(&name)}

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

<table>
<thead>
<tr>
<th>name</th>
<th>the name of the event source variable</th>
</tr>
</thead>
</table>

7.23.2.4  #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

<table>
<thead>
<tr>
<th>name</th>
<th>the name of the event source variable</th>
</tr>
</thead>
</table>

7.23.3  Typedef Documentation

7.23.3.1  typedef struct event_source event_source_t

Event Source structure.
7.23 Event Flags

7.23.3.2 typedef void(*evhandler_t)(eventid_t id)

Event Handler callback function.

7.23.4 Function Documentation

7.23.4.1 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

| 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:

Normal API, 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
 chSysLock
   _stats_start_measure
     _crit_thd
   _dbg_check_lock
   chEvtRegisterMaskWithFlags
    chSysUnlock
      _dbg_check_unlock
      _stats_stop_measure
        _crit_thd
```

7.23.4.2 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

| In | esp | pointer to the event_source_t structure |
| In | elp | pointer to the event_listener_t structure |

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

Here is the call graph for this function:

```
 chipEvUnregister
   \_stats_start_measure
   \_crit_thd
   \_dbg_check_lock
   chSysLock
   chEvUnregister
   \_dbg_check_unlock
   chSysUnlock
   \_stats_stop_measure
   \_crit_thd
```

7.23.4.3 eventmask_t chEvGetAndClearEvents ( eventmask_t events )
Clears the pending events specified in the events mask.

Parameters

| In | events | the events to be cleared |

Returns
The mask of pending events that were cleared.

Function Class:
This is an I-Class API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.
7.23.4.4 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:

```
chSysLock
  chEvtGetAndClearEvents
  chEvtGetAndClearEventsI
  chSysUnlock
  _stats_start_measure
  _crit_thd
  _dbg_check_lock
  _dbg_check_unlock
  _stats_stop_measure
  _crit_thd
```

7.23.4.5 eventmask_t chEvtAddEvents ( eventmask_t events )

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

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:

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

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

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

Postcondition

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

Parameters

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

Function Class:

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

![Call Graph for chEvtGetAndClearFlags](image)

### 7.23.4.7 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:

![Call Graph for chEvtSignal](image)

### 7.23.4.8 void chEvtSignal (thread_t *tp, eventmask_t events)

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

<table>
<thead>
<tr>
<th>in</th>
<th>tp</th>
<th>the thread to be signaled</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>events</td>
<td>the events set to be ORed</td>
</tr>
</tbody>
</table>

Function Class:

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

Here is the call graph for this function:

7.23.4.9 void chEvtSignal ( thread_t *tp, eventmask_t events )

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

Postcondition

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

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>tp</th>
<th>the thread to be signaled</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>events</td>
<td>the events set to be ORed</td>
</tr>
</tbody>
</table>

Function Class:

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

```
chEvtSignalI
   chDbgCheckClassI
   chSchReadyI
   chSysHalt
```

### 7.23.4.10 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:

```
chEvtBroadcastFlags
   chSysLock
   _stats_start_measure
   _crit_thd
   _dbg_check_lock
   chEvtSignalI
   chDbgCheckClassI
   chSchRescheduleS
   _dbg_check_unlock
   _stats_stop_measure
   _crit_thd
```

### 7.23.4.11 eventflags_t chEvtGetAndClearFlagsI (event_listener_t * elp)

Returns the unmasked flags associated to an event_listener_t.
The flags are returned and the event_listener_t flags mask is cleared.

Parameters

| in elp | pointer to the event_listener_t structure |

Returns

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

Function Class:

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

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

Invokes the event handlers associated to an event flags mask.

Parameters

| in events | mask of events to be dispatched |
| in handlers | an array of evhandler_t. The array must have size equal to the number of bits in eventmask_t. |

Function Class:

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

7.23.4.13 eventmask_t chEvtWaitOne ( eventmask_t events )

Waits for exactly one of the specified events.

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

Note

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

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

Parameters

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

Returns

The mask of the lowest event id served and cleared.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.
7.23.4.14 \texttt{eventmask\_t chEvtWaitAny ( eventmask\_t events )}

Waits for any of the specified events.

The function waits for any event among those specified in \texttt{events} to become pending then the events are cleared and returned.

Parameters

\begin{itemize}
  \item \texttt{in events} events that the function should wait for, \texttt{ALL\_EVENTS} enables all the events
\end{itemize}

Returns

The mask of the served and cleared events.

Function Class:

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

```
7.23.4.15  eventmask_t chEvtWaitAll ( eventmask_t events )

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

Parameters

- **in** events events that the function should wait for, **ALL_EVENTS** requires all the events

Returns

- The mask of the served and cleared events.

Function Class:

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

```
<table>
<thead>
<tr>
<th>Function</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>chEvtWaitAll</td>
<td></td>
</tr>
<tr>
<td>chSchGoSleepS</td>
<td></td>
</tr>
<tr>
<td>chSysLock</td>
<td></td>
</tr>
<tr>
<td>chSysUnlock</td>
<td></td>
</tr>
<tr>
<td>_stats_start_measure</td>
<td></td>
</tr>
<tr>
<td>_crit_thd</td>
<td></td>
</tr>
<tr>
<td>_dbg_check_lock</td>
<td></td>
</tr>
<tr>
<td>chDbgCheckClassS</td>
<td></td>
</tr>
<tr>
<td>queue_fifo_remove</td>
<td></td>
</tr>
<tr>
<td>chSysLockFromISR</td>
<td></td>
</tr>
<tr>
<td>chSysUnlockFromISR</td>
<td></td>
</tr>
<tr>
<td>chSemFastSignalI</td>
<td></td>
</tr>
<tr>
<td>queue_dequeue</td>
<td></td>
</tr>
<tr>
<td>chSchReadyI</td>
<td></td>
</tr>
<tr>
<td>_dbg_check_unlock</td>
<td></td>
</tr>
<tr>
<td>_stats_stop_measure</td>
<td></td>
</tr>
<tr>
<td>_crit_thd</td>
<td></td>
</tr>
</tbody>
</table>
```

7.23.4.16 `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**

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

ChibiOS/RT
Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>timeout</th>
<th>the number of ticks before the operation timeouts, the following special values are allowed:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>• <strong>TIME_IMMEDIATE</strong> immediate timeout.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <strong>TIME_INFINITY</strong> 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:

7.23.4.17  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

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

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

Returns

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:

7.23.4.18 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

| in | events | events that the function should wait for, ALL_EVENTS requires all the events |
### Parameters

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

### Returns

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:

![Call Graph](call_graph.png)

### 7.23.4.19 static void chEvtObjectInit ( event_source_t * esp ) [inline],[static]

Initializes an Event Source.

**Note**

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

| in  | esp | pointer to the event_source_t structure |

Function Class:

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

7.23.4.20 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

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

Function Class:

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

Here is the call graph for this function:

![Call Graph](image)

7.23.4.21 static void chEvtRegister ( event_source_t * esp, event_listener_t * elp, eventid_t event )

[inline], [static]

Registers an Event Listener on an Event Source.

Note

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

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

**Function Class:**

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

Here is the call graph for this function:

```plaintext
chEvtRegister ── chEvtRegisterMask ── chEvtRegisterMaskWithFlags
```

7.23.4.22 static bool chEvtIsListeningI ( event_source_t *esp ) [inline],[static]

Verifies if there is at least one event_listener_t registered.

**Parameters**

| in  | esp | pointer to the event_source_t structure |

**Returns**

The event source status.

**Function Class:**

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

7.23.4.23 static void chEvtBroadcast ( event_source_t *esp ) [inline],[static]

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

**Parameters**

| in  | esp | pointer to the event_source_t structure |

**Function Class:**

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

```
chEvtBroadcast chEvtBroadcastFlags
chSysLock
chEvtBroadcastFlagsI
chSchRescheduleS
chSysUnlock
```

### 7.23.4.24 static void chEvtBroadcastI (event_source_t *esp) [inline],[static]

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

**Postcondition**

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

**Parameters**

- *in* esp pointer to the event_source_t structure

**Function Class:**

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

Here is the call graph for this function:

```
chEvtBroadcast chEvtBroadcastFlags chDbgCheckClassI
chEvtSignal
```

### 7.23.4.25 static eventmask_t chEvtAddEventsI (eventmask_t events) [inline],[static]

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

| in | events | the events to be added |

Returns

The mask of currently pending events.

Function Class:

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

\texttt{7.23.4.26 static eventmask_t chEvtGetEventsX ( void ) [inline],[static]}

Returns the events mask.

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

Returns

The pending events mask.

Function Class:

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

7.24.1 Detailed Description

Synchronous inter-thread messages APIs and services.

Operation Mode

Synchronous messages are an easy to use and fast IPC mechanism, threads can both act as message servers and/or message clients, the mechanism allows data to be carried in both directions. Note that messages are not copied between the client and server threads but just a pointer passed so the exchange is very time efficient. Messages are scalar data types of type msg_t that are guaranteed to be size compatible with data pointers. Note that on some architectures function pointers can be larger that msg_t. Messages are usually processed in FIFO order but it is possible to process them in priority order by enabling the CH_CFG_USE_MESSAGES_PRIORITY option in chconf.h.

Precondition

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

Postcondition

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

Functions

- msg_t chMsgSend (thread_t *tp, msg_t msg)
  Sends a message to the specified thread.
- thread_t * chMsgWait (void)
  Suspends the thread and waits for an incoming message.
- void chMsgRelease (thread_t *tp, msg_t msg)
  Releases a sender thread specifying a response message.
- static bool chMsgIsPendingI (thread_t *tp)
  Evaluate 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.

7.24.2 Function Documentation

7.24.2.1 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:

```
    _stats_start_measure
        _ent_thd
    _ent_lock
        _dbg_check_lock
    chSchReady
        chDbgCheckClassS
            queue_fifo_remove
    chSchGoSleepS
        chSemFastSignalI
            chSysLockFromISR
            chSysUnlockFromISR
            queue_dequeue
    chSysLock
        _dbg_check_unlock
    chSysUnlock
        _stats_stop_measure
        _ent_thd
```

7.24.2.2  thread_t *chMsgWait ( void )

Suspends the thread and waits for an incoming message.

Postcondition

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

Note

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

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

Returns

A pointer to the thread carrying the message.

Function Class:

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

Here is the call graph for this function:

```
7.24.2.3 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]

7.24.2.4 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:

![Call Graph]
7.24.2.5 static msg_t chMsgGet ( thread_t * tp ) [inline], [static]

Returns the message carried by the specified thread.

Precondition

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

Parameters

| in | tp | pointer to the thread |

Returns

The message carried by the sender.

Function Class:

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

7.24.2.6 static void chMsgReleaseS ( thread_t * tp, msg_t msg ) [inline], [static]

Releases the thread waiting on top of the messages queue.

Precondition

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

Parameters

| in | tp | pointer to the thread |
| in | msg | message to be returned to the sender |

Function Class:

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

Here is the call graph for this function:
7.25 Dynamic Threads

7.25.1 Detailed Description

Dynamic threads related APIs and services.

Functions

- `thread_t * chThdCreateFromHeap (memory_heap_t *heapp, size_t size, const char *name, tprio_t prio, tfunc_t pf, void *arg)`
  
  Creates a new thread allocating the memory from the heap.

- `thread_t * chThdCreateFromMemoryPool (memory_pool_t *mp, const char *name, tprio_t prio, tfunc_t pf, void *arg)`

  Creates a new thread allocating the memory from the specified memory pool.

7.25.2 Function Documentation

7.25.2.1 `thread_t * chThdCreateFromHeap (memory_heap_t *heapp, size_t size, const char *name, tprio_t prio, tfunc_t pf, void *arg)`

Creates a new thread allocating the memory from the heap.

Precondition

The configuration options `CH_CFG_USE_DYNAMIC` and `CH_CFG_USE_HEAP` must be enabled in order to use this function.

Note

A thread can terminate by calling `chThdExit()` or by simply returning from its main function. The memory allocated for the thread is not released automatically; it is the responsibility of the creator thread to call `chThdWait()` and then release the allocated memory.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>heapp</th>
<th>heap from which allocate the memory or NULL for the default heap</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>size</td>
<td>size of the working area to be allocated</td>
</tr>
<tr>
<td>in</td>
<td>name</td>
<td>thread name</td>
</tr>
<tr>
<td>in</td>
<td>prio</td>
<td>the priority level for the new thread</td>
</tr>
<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 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.
7.25 Dynamic Threads

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

Here is the call graph for this function:

![Call Graph Image]

7.25.2.2 thread_t * chThdCreateFromMemoryPool ( memory_pool_t * mp, const char * name, tprio_t prio, tfunc_t pf, void * arg )

Creates a new thread allocating the memory from the specified memory pool.

Precondition

The configuration options CH_CFG_USE_DYNAMIC and CH_CFG_USE_MEMPOOLS must be enabled in order to use this function.
The pool must be initialized to contain only objects with alignment PORT_WORKING_AREA_ALIGN.

Note

A thread can terminate by calling chThdExit() or by simply returning from its main function.
The memory allocated for the thread is not released automatically, it is responsibility of the creator thread to call chThdWait() and then release the allocated memory.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>mp</th>
<th>pointer to the memory pool object</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>name</td>
<td>thread name</td>
</tr>
<tr>
<td>in</td>
<td>prio</td>
<td>the priority level for the new thread</td>
</tr>
<tr>
<td>in</td>
<td>pf</td>
<td>the thread function</td>
</tr>
<tr>
<td>in</td>
<td>arg</td>
<td>an argument passed to the thread function. It can be NULL.</td>
</tr>
</tbody>
</table>
Returns

The pointer to the 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:
7.26 Registry

7.26.1 Detailed Description

Threads Registry related APIs and services.

Operation mode

The Threads Registry is a double linked list that holds all the active threads in the system. Operations defined for the registry:

• First, returns the first, in creation order, active thread in the system.
• Next, returns the next, in creation order, active thread in the system.

The registry is meant to be mainly a debug feature, for example, using the registry a debugger can enumerate the active threads in any given moment or the shell can print the active threads and their state. Another possible use is for centralized threads memory management, terminating threads can pulse an event source and an event handler can perform a scansion of the registry in order to recover the memory.

Precondition

In order to use the threads registry the `CH_CFG_USE_REGISTRY` option must be enabled in `chconf.h`.

Macros

• #define REG_REMOVE(tp)
  Removes a thread from the registry list.
• #define REG_INSERT(tp)
  Adds a thread to the registry list.

Data Structures

• struct chdebug_t
  ChibiOS/RT memory signature record.

Functions

• thread_t * chRegFirstThread (void)
  Returns the first thread in the system.
• thread_t * chRegNextThread (thread_t *tp)
  Returns the thread next to the specified one.
• thread_t * chRegFindThreadByName (const char *name)
  Retrieves a thread pointer by name.
• thread_t * chRegFindThreadByPointer (thread_t *tp)
  Confirms that a pointer is a valid thread pointer.
• thread_t * chRegFindThreadByWorkingArea (stkalign_t *wa)
  Confirms that a working area is being used by some active thread.
• static void chRegSetThreadName (const char *name)
  Sets the current thread name.
• static const char * chRegGetNameX (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.
7.26.2 Macro Definition Documentation

7.26.2.1 #define REG_REMOVE( tp )

Value:

```
(tpe)->older->newer = (tp)->newer;
(tp)->newer->older = (tp)->older;
```

Removes a thread from the registry list.

Note
This macro is not meant for use in application code.

Parameters

| in | tp | thread to remove from the registry |

7.26.2.2 #define REG_INSERT( tp )

Value:

```
(tpe)->newer = (thread_t *)&ch.rlist;
(tp)->older = ch.rlist.older;
(tp)->older->newer = (tp);
ch.rlist.older = (tp);
```

Adds a thread to the registry list.

Note
This macro is not meant for use in application code.

Parameters

| in | tp | thread to add to the registry |

7.26.3 Function Documentation

7.26.3.1 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.
7.26 Registry

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:

7.26.3.2 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
in 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:

### 7.26.3.3 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:

![Call Graph](image)

### 7.26.3.4 `thread_t * chFindThreadByPointer ( thread_t * tp )`

Confirms that a pointer is a valid thread pointer.

**Note**

The reference counter of the found thread is increased by one so it cannot be disposed incidentally after the pointer has been returned.

**Parameters**

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

**Returns**

A pointer to the found thread.

**Return values**

- `NULL` if a matching thread has not been found.

**Function Class:**

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

Here is the call graph for this function:

![Call Graph](image)
7.26.3.5  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:

<table>
<thead>
<tr>
<th>chRegFindThread</th>
<th>chRegNextThread</th>
<th>chSysUnlock</th>
<th>chThdRelease</th>
</tr>
</thead>
<tbody>
<tr>
<td>chRegFindThreadByWorkingArea</td>
<td>chRegNextThread</td>
<td>chSysLock</td>
<td>chThdGetWorkingAreaX</td>
</tr>
</tbody>
</table>

7.26.3.6  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 |

ChibiOS/RT
Function Class:

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

7.26.3.7 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

\begin{verbatim}
in tp pointer to the thread\end{verbatim}

Returns

Thread name as a zero terminated string.

Return values

\begin{verbatim}
NULL if the thread name has not been set\end{verbatim}

7.26.3.8 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

\begin{verbatim}
in tp pointer to the thread
in name thread name as a zero terminated string\end{verbatim}

Function Class:

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

7.27.1  Detailed Description

Modules

- Checks and Assertions
- Tracing
- Statistics
7.28 Checks and Assertions

7.28.1 Detailed Description

Debug APIs and services:

- Runtime system state and call protocol check. The following panic messages can be generated:
  - SV#1, misplaced `chSysDisable()`.
    - Called from an ISR.
    - Called from a critical zone.
  - SV#2, misplaced `chSysSuspend()`.
    - Called from an ISR.
    - Called from a critical zone.
  - SV#3, misplaced `chSysEnable()`.
    - Called from an ISR.
    - Called from a critical zone.
  - SV#4, misplaced `chSysLock()`.
    - Called from an ISR.
    - Called from a critical zone.
  - SV#5, misplaced `chSysUnlock()`.
    - Called from an ISR.
    - Not called from a critical zone.
  - SV#6, misplaced `chSysLockFromISR()`.
    - Not called from an ISR.
    - Called from a critical zone.
  - SV#7, misplaced `chSysUnlockFromISR()`.
    - Not called from an ISR.
    - Not called from a critical zone.
  - SV#8, misplaced `CH_IRQ_PROLOGUE()`.
    - Not called at ISR begin.
    - Called from a critical zone.
  - SV#9, misplaced `CH_IRQ_EPILOGUE()`.
    - `CH_IRQ_PROLOGUE()` missing.
    - Not called at ISR end.
    - Called from a critical zone.
  - SV#10, misplaced I-class function.
    - I-class function not called from within a critical zone.
  - SV#11, misplaced S-class function.
    - S-class function not called from within a critical zone.
    - Called from an ISR.

- Parameters check.
- Kernel assertions.

Note

Stack checks are not implemented in this module but in the port layer in an architecture-dependent way.
Debug related settings

- #define CH_DBG_STACK_FILL_VALUE 0x55
  Fill value for thread stack area in debug mode.

Macro Functions

- #define chDbgCheck(c)
  Function parameters check.
- #define chDbgAssert(c, r)
  Condition assertion.

Functions

- void _dbg_check_disable (void)
  Guard code for chSysDisable().
- void _dbg_check_suspend (void)
  Guard code for chSysSuspend().
- void _dbg_check_enable (void)
  Guard code for chSysEnable().
- void _dbg_check_lock (void)
  Guard code for chSysLock().
- void _dbg_check_unlock (void)
  Guard code for chSysUnlock().
- void _dbg_check_lock_from_isr (void)
  Guard code for chSysLockFromIsr().
- void _dbg_check_unlock_from_isr (void)
  Guard code for chSysUnlockFromIsr().
- void _dbg_check_enter_isr (void)
  Guard code for CH_IRQ_PROLOGUE().
- void _dbg_check_leave_isr (void)
  Guard code for CH_IRQ_EPILOGUE().
- void chDbgCheckClassI (void)
  I-class functions context check.
- void chDbgCheckClassS (void)
  S-class functions context check.

7.28.2 Macro Definition Documentation

7.28.2.1 #define CH_DBG_STACK_FILL_VALUE 0x55
Fill value for thread stack area in debug mode.

7.28.2.2 #define chDbgCheck(c)

Value:
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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>c</code></td>
<td>the condition to be verified to be true</td>
</tr>
</tbody>
</table>

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

### 7.28.2.3 `#define chDbgAssert( c, r )`

**Value:**

Function 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><code>c</code></td>
<td>the condition to be verified to be true</td>
</tr>
<tr>
<td><code>r</code></td>
<td>a remark string</td>
</tr>
</tbody>
</table>

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

7.28.3.1 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:

7.28.3.2 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:

7.28.3.3 void _dbg_check_enable ( void )

Guard code for chSysEnable().

Function Class:

Not an API, this function is for internal use only.
7.28 Checks and Assertions

Here is the call graph for this function:

```
_dbg_check_enable                chSysHalt                _trace_halt
```

7.28.3.4  void _dbg_check_lock ( void )

Guard code for chSysLock().

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

Here is the call graph for this function:

```
_dbg_check_lock                chSysHalt                _trace_halt
```

7.28.3.5  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
```
7.28.3.6 void _dbg_check_lock_from_isr ( void )

Guard code for chSysLockFromIsr().

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

Here is the call graph for this function:

[Call graph diagram]

7.28.3.7 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 diagram]

7.28.3.8 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 diagram]
7.28 Checks and Assertions

### 7.28.3.9 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 for _dbg_check_leave_isr](chart.png)

### 7.28.3.10 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 for chDbgCheckClassI](chart.png)

### 7.28.3.11 void chDbgCheckClassS ( void )

S-class functions context check.

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

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

![Call Graph Diagram]
7.29 Tracing

7.29.1 Detailed Description

System events tracing service.

Trace record types

- `#define CH_TRACE_TYPE_UNUSED 0U`
- `#define CH_TRACE_TYPE_SWITCH 1U`
- `#define CH_TRACE_TYPE_ISR_ENTER 2U`
- `#define CH_TRACE_TYPE_ISR_LEAVE 3U`
- `#define CH_TRACE_TYPE_HALT 4U`
- `#define CH_TRACE_TYPE_USER 5U`

Events to trace

- `#define CH_DBG_TRACE_MASK_DISABLED 255U`
- `#define CH_DBG_TRACE_MASK_NONE 0U`
- `#define CH_DBG_TRACE_MASK_SWITCH 1U`
- `#define CH_DBG_TRACE_MASK_ISR 2U`
- `#define CH_DBG_TRACE_MASK_HALT 4U`
- `#define CH_DBG_TRACE_MASK_USER 8U`
- `#define CH_DBG_TRACE_MASK_SLOW`
- `#define CH_DBG_TRACE_MASK_ALL`

Debug related settings

- `#define CH_DBG_TRACE_MASK CH_DBG_TRACE_MASK_DISABLED`
  
  Trace buffer entries.
- `#define CH_DBG_TRACE_BUFFER_SIZE 128`
  
  Trace buffer entries.

Data Structures

- `struct ch_trace_event_t`
  
  Trace buffer record.
- `struct ch_trace_buffer_t`
  
  Trace buffer header.

Functions

- `static NOINLINE void trace_next (void)`
  
  Writes a time stamp and increases the trace buffer pointer.
- `void _trace_init (void)`
  
  Trace circular buffer subsystem initialization.
- `void _trace_switch (thread_t *ntp, thread_t *otp)`
  
  Inserts in the circular debug trace buffer a context switch record.
- `void _trace_isr_enter (const char *isr)`
  
  Inserts in the circular debug trace buffer an ISR-enter record.
• void _trace_isr_leave (const char *isr)
    Inserts in the circular debug trace buffer an ISR-leave record.
• void _trace_halt (const char *reason)
    Inserts in the circular debug trace buffer an halt record.
• void chDbgWriteTrace1 (void *up1, void *up2)
    Adds an user trace record to the trace buffer.
• void chDbgWriteTrace (void *up1, void *up2)
    Adds an user trace record to the trace buffer.
• void chDbgSuspendTraceI (uint16_t mask)
    Suspends one or more trace events.
• void chDbgSuspendTrace (uint16_t mask)
    Suspends one or more trace events.
• void chDbgResumeTraceI (uint16_t mask)
    Resumes one or more trace events.
• void chDbgResumeTrace (uint16_t mask)
    Resumes one or more trace events.

7.29.2 Macro Definition Documentation

7.29.2.1 #define CH_DBG_TRACE_MASK CH_DBG_TRACE_MASK_DISABLED
Trace buffer entries.

7.29.2.2 #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.

7.29.3 Function Documentation

7.29.3.1 static NOINLINE void trace_next ( void ) [static]
Writes a time stamp and increases the trace buffer pointer.

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

Here is the call graph for this function:
7.29.3.2 void _trace_init ( void )

Trace circular buffer subsystem initialization.

Note

Internal use only.

7.29.3.3 void _trace_switch ( thread_t * ntp, thread_t * otp )

Inserts in the circular debug trace buffer a context switch record.

Parameters

in ntp the thread being switched in
in otp the thread being switched out

Function Class:

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

Here is the call graph for this function:

```
__trace_switch -> trace_next -> chVTGetSystemTimeX
```

7.29.3.4 void _trace_isr_enter ( const char * isr )

Inserts in the circular debug trace buffer an ISR-enter record.

Parameters

in isr name of the isr

Function Class:

Not an API, this function is for internal use only.
Here is the call graph for this function:

```
_trace_isr_enter -> trace_next -> chVTGetSystemTimeX
```

### 7.29.3.5 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.

Here is the call graph for this function:

```
_trace_isr_leave -> trace_next -> chVTGetSystemTimeX
```

### 7.29.3.6 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.
Here is the call graph for this function:

7.29.3.7  void chDbgWriteTraceI ( 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:

7.29.8  void chDbgWriteTrace ( 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:

```
.chDbgWriteTrace
.chSysLock
.chDbgWriteTraceI
.chSysUnlock
/stats_start_measure
/crit_thd
_dbg_check_lock
.chDbgCheckClassI
.trace_next
_dbg_check_unlock
/stats_stop_measure
/crit_thd
```

### 7.29.3.9 `void chDbgSuspendTraceI ( 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:

```
.chDbgSuspendTraceI
.chDbgCheckClassI
.chSysHalt
```

### 7.29.3.10 `void chDbgSuspendTrace ( uint16_t mask )`

Suspends one or more trace events.
### 7.29.3.11 void chDbgResumeTraceI (uint16_t mask)

Resumes one or more trace events.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in mask</td>
<td>mask of the trace events to be resumed</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.

**Call Graph:**

```
chDbgResumeTraceI  
<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>chDbgCheckClassI</td>
</tr>
<tr>
<td>chSysHalt</td>
</tr>
</tbody>
</table>
```

---

#### 7.29 Tracing Parameters

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in 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.

**Call Graph:**

```
chDbgSuspendTrace  
<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>chSysLock</td>
</tr>
<tr>
<td>chSysUnlock</td>
</tr>
<tr>
<td>_stats_start_measure</td>
</tr>
<tr>
<td>_crit_thd</td>
</tr>
<tr>
<td>_dbg_check_lock</td>
</tr>
<tr>
<td>_dbg_check_unlock</td>
</tr>
<tr>
<td>_stats_stop_measure</td>
</tr>
<tr>
<td>_crit_thd</td>
</tr>
</tbody>
</table>
```
7.29.3.12  void chDbgResumeTrace ( uint16_t mask )

Resumes one or more trace events.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mask</td>
<td>mask of the trace events to be resumed</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:
7.30 Statistics

7.30.1 Detailed Description

Statistics services.

Data Structures

- struct kernel_stats_t
  
  Type of a kernel statistics structure.

Functions

- void _stats_init (void)
  
  Initializes the statistics module.

- void _stats_increase_irq (void)
  
  Increases the IRQ counter.

- void _stats_ctxswc (thread_t *ntp, thread_t *otp)
  
  Updates context switch related statistics.

- void _stats_start_measure_crit_thd (void)
  
  Starts the measurement of a thread critical zone.

- void _stats_stop_measure_crit_thd (void)
  
  Stops the measurement of a thread critical zone.

- void _stats_start_measure_crit_isr (void)
  
  Starts the measurement of an ISR critical zone.

- void _stats_stop_measureCrit_isr (void)
  
  Stops the measurement of an ISR critical zone.

7.30.2 Function Documentation

7.30.2.1 void _stats_init ( void )

Initializes the statistics module.

Function Class:

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

Here is the call graph for this function:

![Call Graph](chart.png)
7.30.2.2  void _stats_increase_irq ( void )

Increases the IRQ counter.

7.30.2.3  void _stats_ctxswc ( thread_t * ntp, thread_t * otp )

Updates context switch related statistics.

Parameters

<table>
<thead>
<tr>
<th></th>
<th>ntp</th>
<th>the thread to be switched in</th>
</tr>
</thead>
<tbody>
<tr>
<td></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 for _stats_ctxswc](image)

7.30.2.4  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 for _stats_start_measure_crit_thd](image)

7.30.2.5  void _stats_stop_measure_crit_thd ( void )

Stops the measurement of a thread critical zone.
Here is the call graph for this function:

7.30.2.6  void _stats_start_measure_crit_isr ( void )

Starts the measurement of an ISR critical zone.

Here is the call graph for this function:

7.30.2.7  void _stats_stop_measure_crit_isr ( void )

Stops the measurement of an ISR critical zone.

Here is the call graph for this function:
7.31 OS Library

7.31.1 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
7.32 Version Numbers and Identification

7.32.1 Detailed Description

OS Library related info.

Macros

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

ChibiOS/LIB version identification

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

7.32.2 Macro Definition Documentation

7.32.2.1 #define _CHIBIOS_OSLIB_

ChibiOS/LIB identification macro.

7.32.2.2 #define CH_OSLIB_STABLE 1

Stable release flag.

7.32.2.3 #define CH_OSLIB_VERSION "1.1.2"

OS Library version string.

7.32.2.4 #define CH_OSLIB_MAJOR 1

OS Library version major number.

7.32.2.5 #define CH_OSLIB_MINOR 1

OS Library version minor number.

7.32.2.6 #define CH_OSLIB_PATCH 2

OS Library version patch number.
7.33 Synchronization

7.33.1 Detailed Description

Synchronization services.

Modules

- Binary Semaphores
- Mailboxes
- Pipes
7.34 Binary Semaphores

7.34.1 Detailed Description

Macros

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

- 
  ```c
  #define BSEMAPHORE_DECL(name, taken) binary_semaphore_t name = _BSEMAPHORE_DATA(name, taken)
  ```
  Static semaphore initializer.

Typedefs

- 
  ```c
  typedef struct ch_binary_semaphore binary_semaphore_t
  ```
  Binary semaphore type.

Data Structures

- 
  ```c
  struct ch_binary_semaphore
  ```
  Binary semaphore type.

Functions

- 
  ```c
  static void chBSemObjectInit (binary_semaphore_t *bsp, bool taken)
  ```
  Initializes a binary semaphore.

- 
  ```c
  static msg_t chBSemWait (binary_semaphore_t *bsp)
  ```
  Wait operation on the binary semaphore.

- 
  ```c
  static msg_t chBSemWaitS (binary_semaphore_t *bsp)
  ```
  Wait operation on the binary semaphore.

- 
  ```c
  static msg_t chBSemWaitTimeoutS (binary_semaphore_t *bsp, sysinterval_t timeout)
  ```
  Wait operation on the binary semaphore.

- 
  ```c
  static msg_t chBSemWaitTimeout (binary_semaphore_t *bsp, sysinterval_t timeout)
  ```
  Wait operation on the binary semaphore.

- 
  ```c
  static void chBSemResetI (binary_semaphore_t *bsp, bool taken)
  ```
  Reset operation on the binary semaphore.

- 
  ```c
  static void chBSemReset (binary_semaphore_t *bsp, bool taken)
  ```
  Reset operation on the binary semaphore.

- 
  ```c
  static void chBSemSignalI (binary_semaphore_t *bsp)
  ```
  Performs a signal operation on a binary semaphore.

- 
  ```c
  static void chBSemSignal (binary_semaphore_t *bsp)
  ```
  Performs a signal operation on a binary semaphore.

- 
  ```c
  static bool chBSemGetStateI (const binary_semaphore_t *bsp)
  ```
  Returns the binary semaphore current state.

7.34.2 Macro Definition Documentation

7.34.2.1

```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>

#### 7.34.2.2 `#define BSEMAPHORE_DECL( name, taken )` binary_semaphore_t name = _BSEMAPHORE_DATA(name, taken)

Static semaphore initializer.

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

#### Parameters

<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>

#### 7.34.3 Typedef Documentation

##### 7.34.3.1 `typedef struct ch_binary_semaphore binary_semaphore_t`

Binary semaphore type.

#### 7.34.4 Function Documentation

##### 7.34.4.1 `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:
7.34 Binary Semaphores

7.34.2 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

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

Function Class:

Normal API, 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 ↔ bSysLock ↔ bSemWaitS ↔ bSysUnlock
```

7.34.3 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
chSemWaitS chSysHalt
queue_isempty queue_notempty
chSchGoSleepS
```

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

Wait operation on the binary semaphore.

Parameters

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

Returns

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

Return values

<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>
<tr>
<td>MSG_TIMEOUT</td>
<td>if the binary semaphore has not been signaled or reset within the specified timeout.</td>
</tr>
</tbody>
</table>

Function Class:

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]

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

Wait operation on the binary semaphore.

Parameters

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

Returns

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

Return values

| MSG_OK | if the binary semaphore has been successfully taken. |
| MSG_RESET | if the binary semaphore has been reset using bsemReset(). |
| MSG_TIMEOUT | if the binary semaphore has not been signaled or reset within the specified timeout. |

Function Class:

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

```
chBSemWaitTimeout -> chSemWaitTimeout
|                | chSemWaitTimeoutS
|                | chSysUnlock
| chBSemWaitTimeout |
```

### 7.34.4.6 static void chBSemResetI (binary_semaphore_t *bsp, bool taken) [inline], [static]

Reset operation on the binary semaphore.

**Note**

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

This function does not reschedule.

**Parameters**

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

**Function Class:**

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

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

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

Reset operation on the binary semaphore.

Note

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

Parameters

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

Function Class:

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

Here is the call graph for this function:

![Call Graph](image)

7.34.4.8 static void chSemSignal (binary_semaphore_t * bsp) [inline],[static]

Performs a signal operation on a binary semaphore.

Note

This function does not reschedule.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>bsp</th>
<th>pointer to a binary_semaphore_t structure</th>
</tr>
</thead>
</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)

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

Performs a signal operation on a binary semaphore.

**Parameters**

| **in bsp** | pointer to a binary_semaphore_t structure |

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.
7.34.4.10 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:
7.35 Mailboxes

7.35.1 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)`
- `msg_t chMBPostI(mailbox_t *mbp, msg_t msg)`
- `msg_t chMBPostAheadTimeout(mailbox_t *mbp, msg_t msg, sysinterval_t timeout)`
- `msg_t chMBPostAheadTimeoutS(mailbox_t *mbp, msg_t msg, sysinterval_t timeout)`
- `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)`
- `msg_t chMBFetchTimeoutS(mailbox_t *mbp, msg_t *msgp, sysinterval_t timeout)`
- `msg_t chMBFetchI(mailbox_t *mbp, msg_t *msgp)`

Retrieves a message from a mailbox.

- `static size_t chMBGetSizeI(const mailbox_t *mbp)`
- `static size_t chMBGetUsedCountI(const mailbox_t *mbp)`
- `static size_t chMBGetFreeCountI(const mailbox_t *mbp)`
- `static msg_t chMBPeekI(const mailbox_t *mbp)`
- `static void chMBResumeX(mailbox_t *mbp)`

Terminates the reset state.

### 7.35.2 Macro Definition Documentation

#### 7.35.2.1 

```
#define _MAILBOX_DATA(name, buffer, size) {
    (msg_t *)(buffer),
    (msg_t *)(buffer) + size,
    (msg_t *)(buffer),
    (msg_t *)(buffer),
    (size_t)0,
    false,
    _THREADS_QUEUE_DATA(name.qw),
    _THREADS_QUEUE_DATA(name.qr),
}
```

Data part of a static mailbox initializer.

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

**Value:**

- `name` the name of the mailbox variable
- `buffer` pointer to the mailbox buffer array of `msg_t`
- `size` number of `msg_t` elements in the buffer array
7.35 Mailboxes

7.35.2 #define MAILBOX_DECL( name, buffer, size ) mailbox_t name = _MAILBOX_DATA(name, buffer, size)

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

Parameters

| in  | name     | the name of the mailbox variable |
| in  | buffer   | pointer to the mailbox buffer array of msg_t |
| in  | size     | number of msg_t elements in the buffer array |

7.35.3 Function Documentation

7.35.3.1 void chMObjectInit( mailbox_t * mbp, msg_t * buf, size_t n )

Initializes a mailbox_t object.

Parameters

| out | mbp    | the pointer to the mailbox_t structure to be initialized |
| in  | buf    | pointer to the messages buffer as an array of msg_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.

Here is the call graph for this function:

```
chMObjectInit chThdQueueObjectInit queue_init
```

7.35.3.2 void chMReset( mailbox_t * mbp )

Resets a mailbox_t object.
All the waiting threads are resumed with status MSG_RESET and the queued messages are lost.

Postcondition

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

Parameters

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

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:

```
7.35.3.3 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.
7.35 Mailboxes

Here is the call graph for this function:

![Call Graph]

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

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

Parameters

<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 times out, the following special values are allowed:</td>
</tr>
<tr>
<td></td>
<td>• <code>TIME_IMMEDIATE</code> immediate timeout.</td>
</tr>
<tr>
<td></td>
<td>• <code>TIME_INFINITE</code> no timeout.</td>
</tr>
</tbody>
</table>

Returns

The operation status.

Return values

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

Function Class:

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

ChibiOS/RT
7.35.3.5  msg_t chMBPostTimeoutS ( mailbox_t *mbp, msg_t msg, sysinterval_t timeout )

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

Parameters

<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

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

Function Class:

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

Here is the call graph for this function:

![Call Graph Image]

7.35.3.6 `msg_t chMBPostI ( mailbox_t *mbp, msg_t msg )`

Posts a message into a mailbox.

This variant is non-blocking, the function returns a timeout condition if the queue is full.

Parameters

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

Returns

The operation status.

Return values

- **MSG_OK** if a message has been correctly posted.
- **MSG_RESET** if the mailbox has been reset.
- **MSG_TIMEOUT** if the mailbox is full and the message cannot be posted.

Function Class:

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

Here is the call graph for this function:

![Call Graph Image]
7.35.3.7  msg_t chMBPostAheadTimeout ( mailbox_t * mbp, msg_t msg, sysinterval_t timeout )

Posts an high priority message into a mailbox.
The invoking thread waits until a empty slot in the mailbox becomes available or the specified time runs out.

Parameters

<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:
7.35.3.8  msg_t chMBPostAheadTimeoutS ( mailbox_t *mbp, msg_t msg, sysinterval_t timeout )

Posts an high priority message into a mailbox. The invoking thread waits until an empty slot in the mailbox becomes available or the specified time runs out.

Parameters

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

Returns

The operation status.

Return values

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

Function Class:

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

Here is the call graph for this function:

7.35.3.9  msg_t chMBPostAheadI ( mailbox_t *mbp, msg_t msg )

Posts an high priority message into a mailbox. This variant is non-blocking, the function returns a timeout condition if the queue is full.

Parameters

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

The operation status.

Return values

| 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:

```
msg_t chMBFetchTimeout ( mailbox_t *mbp, msg_t *msgp, sysinterval_t timeout )
```

Retrieves a message from a mailbox.

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

Parameters

| in  | mbp      | the pointer to an initialized mailbox_t object |
| out | msgp     | pointer to a message variable for the received message |
| in  | timeout  | the number of ticks before the operation timeouts, the following special values are allowed: |
|     |          | • TIME_IMMEDIATE immediate timeout. |
|     |          | • TIME_INFINITE no timeout. |

Returns

The operation status.

Return values

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

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

Here is the call graph for this function:

```
7.35.3.11 msg_t chMBFetchTimeoutS ( mailbox_t * mbp, msg_t * msgp, sysinterval_t timeout )

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

Parameters

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

Returns

The operation status.

Return values

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

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

Here is the call graph for this function:

![Call Graph Image]

### 7.35.3.12 msg_t chMBFetchI ( mailbox_t * mbp, msg_t * msgp )

Retrieves a message from a mailbox.

This variant is non-blocking, the function returns a timeout condition if the queue is empty.

#### Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>mbp</th>
<th>the pointer to an initialized mailbox_t object</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>msgp</td>
<td>pointer to a message variable for the received message</td>
</tr>
</tbody>
</table>

#### Returns

The operation status.

#### Return values

<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 → chDbgCheckClassI → chMBGetUsedCountI → chThdDequeueNextI → chSysHalt
```

### 7.35.3.13 static size_t chMBGetSizeI (const mailbox_t *mbp) [inline], [static]

Returns the mailbox buffer size as number of messages.

**Parameters**

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

**Returns**

The size of the mailbox.

**Function Class:**

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

### 7.35.3.14 static size_t chMBGetUsedCountI (const mailbox_t *mbp) [inline], [static]

Returns the number of used message slots into a mailbox.

**Parameters**

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

The number of queued messages.

Function Class:

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

Here is the call graph for this function:

```
chMBGetUsedCountI → chDbgCheckClassI → chSysHalt
```

7.35.3.15 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:

```
chMBGetFreeCountI → chMBGetSizeI → chMBGetUsedCountI → chDbgCheckClassI → chSysHalt
```

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

Returns the next message in the queue without removing it.
7.35 Mailboxes

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
```

7.35.3.17 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.
7.36 Pipes

7.36.1 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.

7.36.2 Macro Definition Documentation

7.36.2.1 #define _PIPE_DATA(name, buffer, size)

Value:

```c
(uint8_t *)[buffer],
(uint8_t *)[buffer] + size,
(uint8_t *)[buffer],
(uint8_t *)[buffer],
(uint8_t *)[buffer],
(size_t)0,
false,
NULL,
NULL,
```
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>

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

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

Parameters

<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>

7.36.3 Function Documentation

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

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

Parameters

<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.

Here is the call graph for this function:

```
pipe_write  chPipeGetFreeCount
            chPipeGetSize
            chPipeGetUsedCount
```

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

Non-blocking pipe read.

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

Parameters

<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.

Here is the call graph for this function:

```
pipe_read  chPipeGetUsedCount
```

ChibiOS/RT
7.36 Pipes

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

Initializes a mailbox_t object.

Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>pp</td>
</tr>
<tr>
<td>in</td>
<td>buf</td>
</tr>
<tr>
<td>in</td>
<td>n</td>
</tr>
</tbody>
</table>

Function Class:

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

7.36.3.4 void chPipeReset ( pipe_t * pp )

Resets a pipe_t object.

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

Postcondition

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

Parameters

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

Function Class:

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

![Call Graph Image]

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

Pipe write with timeout.

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

#### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>pp</code></td>
<td>the pointer to an initialized <code>pipe_t</code> object</td>
</tr>
<tr>
<td><code>bp</code></td>
<td>pointer to the data buffer</td>
</tr>
<tr>
<td><code>n</code></td>
<td>the number of bytes to be written, the value 0 is reserved</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 number of bytes effectively transferred. A number lower than \( n \) means that a timeout occurred or the pipe went in reset state.

Function Class:

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

Here is the call graph for this function:

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

Pipe read with timeout.

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pp</td>
<td>the pointer to an initialized <code>pipe_t</code> object</td>
</tr>
<tr>
<td>bp</td>
<td>pointer to the data buffer</td>
</tr>
<tr>
<td>n</td>
<td>the number of bytes to be read, the value 0 is reserved</td>
</tr>
<tr>
<td>timeout</td>
<td>the number of ticks before the operation timeouts, the following special values are allowed:</td>
</tr>
<tr>
<td></td>
<td>• <code>TIME_IMMEDIATE</code> immediate timeout.</td>
</tr>
<tr>
<td></td>
<td>• <code>TIME_INFINITE</code> no timeout.</td>
</tr>
</tbody>
</table>
Returns

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

Function Class:

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

Here is the call graph for this function:

\[
\text{chPipeReadTimeout} \rightarrow \text{pipe_read} \rightarrow \text{chSysLock} \rightarrow \text{chThdSuspendTimeoutS} \rightarrow \text{chSysUnlock} \rightarrow \text{chThdResume} \rightarrow \text{chPipeGetUsedCount} \rightarrow \text{_stats_start_measure} \rightarrow \text{_crit_thd} \rightarrow \text{_dbg_check_lock} \rightarrow \text{chThdGetSelfX} \rightarrow \text{chSchGoSleepTimeoutS} \rightarrow \text{_dbg_check_unlock} \rightarrow \text{_stats_stop_measure} \rightarrow \text{_crit_thd} \rightarrow \text{chThdResumeS}
\]

7.36.3.7 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.

7.36.3.8 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.

Function Class:

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

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

Returns the number of free byte slots into a pipe.

Parameters

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

Returns

The number of empty byte slots.

Function Class:

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

Here is the call graph for this function:

```
chPipeGetFreeCount
    chPipeGetSize
    chPipeGetUsedCount

chPipeGetFreeCount
```

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

Terminates the reset state.

Parameters

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

Function Class:

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

7.37.1 Detailed Description

Memory Management services.

Modules

- Core Memory Manager
- Memory Heaps
- Memory Pools
7.38 Core Memory Manager

7.38.1 Detailed Description

Core Memory Manager related APIs and services.

Operation mode

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

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

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

By having a centralized memory provider the various allocators can coexist and share the main memory.

This allocator, alone, is also useful for very simple applications that just require a simple way to get memory blocks.

Precondition

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

Note

Compatible with RT and NIL.

Macros

- #define CH_CFG_MEMCORE_SIZE 0

  Managed RAM size.

Typedefs

- typedef void *(+memgetfunc_t)(size_t size, unsigned align)

  Memory get function.

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

  Enhanced memory get function.

Data Structures

- struct memcore_t

  Type of memory core object.

Functions

- void _core_init (void)

  Low level memory manager initialization.

- void * chCoreAllocAlignedWithOffsetI (size_t size, unsigned align, size_t offset)

  Allocates a memory block.
• `void * chCoreAllocAlignedWithOffset (size_t size, unsigned align, size_t offset)`
  Allocates a memory block.
• `size_t chCoreGetStatusX (void)`
  Core memory status.
• `static void * chCoreAllocAlignedI (size_t size, unsigned align)`
  Allocates a memory block.
• `static void * chCoreAllocAligned (size_t size, unsigned align)`
  Allocates a memory block.
• `static void * chCoreAllocI (size_t size)`
  Allocates a memory block.
• `static void * chCoreAlloc (size_t size)`
  Allocates a memory block.

Variables

• `memcore_t ch_memcore`
  Memory core descriptor.

7.38.2 Macro Definition Documentation

7.38.2.1 `#define CH_CFG_MEMCORE_SIZE 0`

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

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

7.38.3 Typedef Documentation

7.38.3.1 `typedef void (*)(size_t size, unsigned align) memgetfunc_t`
Memory get function.

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

7.38.4 Function Documentation

7.38.4.1 `void _core_init (void)`
Low level memory manager initialization.

Function Class:
Not an API, this function is for internal use only.
7.38.4.2 void *chCoreAllocAlignedWithOffsetI ( size_t size, unsigned align, size_t offset )

Allocates a memory block.

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

Parameters

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

Returns

A pointer to the allocated memory block.

Return values

NULL allocation failed, core memory exhausted.

Function Class:

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

Here is the call graph for this function:

7.38.4.3 void *chCoreAllocAlignedWithOffset ( size_t size, unsigned align, size_t offset )

Allocates a memory block.

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

Parameters

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

Returns

A pointer to the allocated memory block.
Return values

| NULL | allocation failed, core memory exhausted. |

Function Class:

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

Here is the call graph for this function:

7.38.4.4 size_t chCoreGetStatusX ( void )

Core memory status.

Returns

The size, in bytes, of the free core memory.

Function Class:

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

7.38.4.5 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.

Parameters

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

Returns

A pointer to the allocated memory block.
Return values

\[
\begin{array}{ll}
\text{NULL} & \text{allocation failed, core memory exhausted.}
\end{array}
\]

Function Class:

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

Here is the call graph for this function:

![Call Graph](image)

7.38.4.6 \texttt{static void∗ chCoreAllocAligned ( size_t size, unsigned align ) [inline],[static]}

Allocates a memory block.

The allocated block is guaranteed to be properly aligned to the specified alignment.

Parameters

\[
\begin{array}{ll}
\text{in} & \text{size} \quad \text{the size of the block to be allocated} \\
\text{in} & \text{align} \quad \text{desired memory alignment}
\end{array}
\]

Returns

A pointer to the allocated memory block.

Return values

\[
\begin{array}{ll}
\text{NULL} & \text{allocation failed, core memory exhausted.}
\end{array}
\]

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.
7.38.4.7 static void *chCoreAlloc ( size_t size ) [inline],[static]

Allocates a memory block.
The allocated block is guaranteed to be properly aligned for a pointer data type.

Parameters

in 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.

Here is the call graph for this function:
7.38.4.8 static void * chCoreAlloc ( size_t size ) [inline],[static]

Allocates a memory block. The allocated block is guaranteed to be properly aligned for a pointer data type.

Parameters

|    | 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.

Here is the call graph for this function:

7.38.5 Variable Documentation

7.38.5.1 memcore_t ch_memcore

Memory core descriptor.
7.39 Memory Heaps

7.39.1 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.

7.39.2 Macro Definition Documentation

7.39.2.1 #define CH_HEAP_ALIGNMENT 8U
Minimum alignment used for heap.

Note
Cannot use the sizeof operator in this macro.

7.39.2.2 #define CH_HEAP_AREA( name, size )

Value:

\[
\text{ALIGNED_VAR(CH_HEAP_ALIGNMENT)}
\text{uint8_t name[MEM_ALIGN_NEXT((size), CH_HEAP_ALIGNMENT)]}
\]
Allocation of an aligned static heap buffer.

7.39.3 Typedef Documentation

7.39.3.1 typedef struct memory_heap memory_heap_t
Type of a memory heap.

7.39.3.2 typedef union heap_header heap_header_t
Type of a memory heap header.

7.39.4 Function Documentation

7.39.4.1 void_heap_init ( void )
Initializes the default heap.

Function Class:
Not an API, this function is for internal use only.
Here is the call graph for this function:

![Call Graph]

7.39.4.2 void chHeapObjectInit ( memory_heap_t *heapp, void *buf, size_t size )

Initializes a memory heap from a static memory area.

Note

The heap buffer base and size are adjusted if the passed buffer is not aligned to CH_HEAP_ALIGNMENT. This mean that the effective heap size can be less than size.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>out heapp</td>
<td>pointer to the memory heap descriptor to be initialized</td>
</tr>
<tr>
<td>in buf</td>
<td>heap buffer base</td>
</tr>
<tr>
<td>in 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.

Here is the call graph for this function:

![Call Graph]

7.39.4.3 void *chHeapAllocAligned ( memory_heap_t *heapp, size_t size, unsigned align )

Allocates a block of memory from the heap by using the first-fit algorithm.
The allocated block is guaranteed to be properly aligned to the specified alignment.

### Parameters
<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td><code>heapp</code></td>
<td>pointer to a heap descriptor or <strong>NULL</strong> in order to access the default heap.</td>
</tr>
<tr>
<td>in</td>
<td><code>size</code></td>
<td>the size of the block to be allocated. Note that the allocated block may be a bit bigger than the requested size for alignment and fragmentation reasons.</td>
</tr>
<tr>
<td>in</td>
<td><code>align</code></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.

#### 7.39.4.4 void chHeapFree ( void *p )

Frees a previously allocated memory block.

### Parameters
<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td><code>p</code></td>
<td>pointer to the memory block to be freed</td>
</tr>
</tbody>
</table>

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

#### 7.39.4.5 size_t chHeapStatus ( memory_heap_t *heapp, size_t *totalp, size_t *largestp )

Reports the heap status.

### Note
This function is meant to be used in the test suite, it should not be really useful for the application code.

### Parameters
<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td><code>heapp</code></td>
<td>pointer to a heap descriptor or <strong>NULL</strong> in order to access the default heap.</td>
</tr>
<tr>
<td>in</td>
<td><code>totalp</code></td>
<td>pointer to a variable that will receive the total fragmented free space or <strong>NULL</strong></td>
</tr>
<tr>
<td>in</td>
<td><code>largestp</code></td>
<td>pointer to a variable that will receive the largest free free block found space or <strong>NULL</strong></td>
</tr>
</tbody>
</table>
Returns

The number of fragments in the heap.

Function Class:

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

7.39.4.6 static void chHeapAlloc ( memory_heap_t * heapp, size_t size ) [inline],[static]

Allocates a block of memory from the heap by using the first-fit algorithm.

The allocated block is guaranteed to be properly aligned for a pointer data type.

Parameters

<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>
</tbody>
</table>

Returns

A pointer to the allocated block.

Return values

| NULL | if the block cannot be allocated. |

Function Class:

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

Here is the call graph for this function:

```
chHeapAlloc ← chHeapAllocAligned
```

7.39.4.7 static size_t chHeapGetSize ( const void * p ) [inline],[static]

Returns the size of an allocated block.

Note

The returned value is the requested size, the real size is the same value aligned to the next CH_HEAP_ALIGNMENT multiple.
7.39 Memory Heaps

Parameters

| in | \( \rho \) | pointer to the memory block |

Returns

Size of the block.

Function Class:

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

7.39.5 Variable Documentation

7.39.5.1 memory_heap_t default_heap [static]

Default heap descriptor.
7.40 Memory Pools

7.40.1 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 *chPoolAllocI (memory_pool_t *mp)`
  
  Allocates an object from a memory pool.
7.40 Memory Pools

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

7.40.2 Macro Definition Documentation

7.40.2.1 #define _MEMORYPOOL_DATA( name, size, align, provider ) {NULL, size, align, provider}

Data part of a static memory pool initializer.

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

Parameters

<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>
Static memory pool initializer.

Statically initialized memory pools require no explicit initialization using `chPoolInit()`.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>name</code></td>
<td>the name of the memory pool variable</td>
</tr>
<tr>
<td><code>size</code></td>
<td>size of the memory pool contained objects</td>
</tr>
<tr>
<td><code>align</code></td>
<td>required memory alignment</td>
</tr>
<tr>
<td><code>provider</code></td>
<td>memory provider function for the memory pool or NULL if the pool is not allowed to grow automatically</td>
</tr>
</tbody>
</table>

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>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>name</code></td>
<td>the name of the memory pool variable</td>
</tr>
<tr>
<td><code>size</code></td>
<td>size of the memory pool contained objects</td>
</tr>
<tr>
<td><code>align</code></td>
<td>required memory alignment</td>
</tr>
</tbody>
</table>

Static guarded memory pool initializer.

Statically initialized guarded memory pools require no explicit initialization using `chGuardedPoolInit()`.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>name</code></td>
<td>the name of the guarded memory pool variable</td>
</tr>
<tr>
<td><code>size</code></td>
<td>size of the memory pool contained objects</td>
</tr>
<tr>
<td><code>align</code></td>
<td>required memory alignment</td>
</tr>
</tbody>
</table>

Initializes an empty memory pool.
7.40 Memory Pools

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>out mp</strong></td>
<td>pointer to a <code>memory_pool_t</code> structure</td>
</tr>
<tr>
<td><strong>in size</strong></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><strong>in align</strong></td>
<td>required memory alignment</td>
</tr>
<tr>
<td><strong>in provider</strong></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.

7.40.3.2 `void chPoolLoadArray ( memory_pool_t * mp, void * p, size_t n )`

Loads a memory pool with an array of static objects.

Precondition

The memory pool must already be initialized.
The array elements must be of the right size for the specified memory pool.
The array elements size must be a multiple of the alignment requirement for the pool.

Postcondition

The memory pool contains the elements of the input array.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>in mp</strong></td>
<td>pointer to a <code>memory_pool_t</code> structure</td>
</tr>
<tr>
<td><strong>in p</strong></td>
<td>pointer to the array first element</td>
</tr>
<tr>
<td><strong>in n</strong></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)

7.40.3.3 `void * chPoolAllocI ( memory_pool_t * mp )`

Allocates an object from a memory pool.
Precondition

The memory pool must already be initialized.

Parameters

| in | mp | pointer to a `memory_pool_t` structure |

Returns

The pointer to the allocated object.

Return values

- **NULL** if pool is empty.

Function Class:

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

Here is the call graph for this function:

```
    chPoolAllocI  chDbgCheckClassI  chSysHalt
```

7.40.3.4  ```
        void ∗ chPoolAlloc ( memory_pool_t ∗ mp )
        ```

Allocates an object from a memory pool.

Precondition

The memory pool must already be initialized.

Parameters

| in | 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:

7.40.3.5 void chPoolFree ( memory_pool_t *mp, void *objp )

Releases an object into a memory pool.

Precondition

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

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>mp</th>
<th>pointer to a memory_pool_t structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>objp</td>
<td>the pointer to the object to be released</td>
</tr>
</tbody>
</table>

Function Class:

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

```
chPoolFreeI -> chDbgCheckClassI -> chSysHalt
```

### 7.40.3.6 void chPoolFree ( memory_pool_t *mp, void *objp )

Releases an object into a memory pool.

**Precondition**

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

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>mp</th>
<th>pointer to a memory_pool_t structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>objp</td>
<td>the pointer to the object to be released</td>
</tr>
</tbody>
</table>

**Function Class:**

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

Here is the call graph for this function:

```
chPoolFree
  ____________________________
  |                          |
  |                           |
  |__________________________|
  |                          |
  |                           |
  |__________________________|
  |                          |
  |                           |
  |__________________________|
  |                          |
  |                           |
  |__________________________|
  |                          |
  |                           |
  |__________________________|
```
7.40 Memory Pools

7.40.3.7 void chGuardedPoolObjectInitAligned ( guarded_memory_pool_t * gmp, size_t size, unsigned align )

Initializes an empty guarded memory pool.

Parameters

| in   | gmp     | pointer to a guarded_memory_pool_t structure |
|      | size    | the size of the objects contained in this guarded memory pool, the minimum accepted size is the size of a pointer to void. |
|      | align   | required memory alignment |

Function Class:

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

Here is the call graph for this function:

```
<table>
<thead>
<tr>
<th>Function Call</th>
</tr>
</thead>
<tbody>
<tr>
<td>chGuardedPoolObjectInitAligned</td>
</tr>
<tr>
<td>chPoolObjectInitAligned</td>
</tr>
<tr>
<td>chSemObjectInit</td>
</tr>
<tr>
<td>queue_init</td>
</tr>
</tbody>
</table>
```

7.40.3.8 void chGuardedPoolLoadArray ( guarded_memory_pool_t * gmp, void * p, size_t n )

Loads a guarded memory pool with an array of static objects.

Precondition

The guarded memory pool must already be initialized.
The array elements must be of the right size for the specified guarded memory pool.

Postcondition

The guarded memory pool contains the elements of the input array.

Parameters

<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:

7.40.3.9 void *chGuardedPoolAllocTimeoutS (guarded_memory_pool_t *gmp, sysinterval_t timeout)

Allocates an object from a guarded memory pool.

**Precondition**

The guarded memory pool must already be initialized.

**Parameters**

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

**Returns**

The pointer to the allocated object.

**Return values**

NULL if the operation timed out.

**Function Class:**

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

Here is the call graph for this function:
7.40 Memory Pools

7.40.3.10 void *chGuardedPoolAllocTimeout ( guarding_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 guarding_memory_pool_t structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>timeout</td>
<td>the number of ticks before the operation timeouts, the following special values are allowed:</td>
</tr>
</tbody>
</table>

- `TIME_IMMEDIATE` immediate timeout.
- `TIME_INFINITE` no timeout.

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:

7.40.3.11 void chGuardedPoolFree ( guarding_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:

---

7.40.3.12 static void chPoolObjectInit (memory_pool_t *mp, size_t size, memgetfunc_t provider ) [inline], [static]

Initializes an empty memory pool.

Parameters

| out | mp | pointer to a memory_pool_t structure |
| in | size | the size of the objects contained in this memory pool, the minimum accepted size is the size of a pointer to void. |
| in | provider | memory provider function for the memory pool or NULL if the pool is not allowed to grow automatically |
Function Class:

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

Here is the call graph for this function:

7.40.3.13\hspace{2em} static void chPoolAdd ( memory_pool_t * mp, void * obj ) \hspace{2em} [inline],[static]

Adds an object to a memory pool.

Precondition

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

Note

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

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mp</td>
<td>pointer to a memory_pool_t structure</td>
</tr>
<tr>
<td>objp</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:
7.40.3.14 static void chPoolAddI ( memory_pool_t * mp, void * objp ) [inline],[static]

Adds an object to a memory pool.

Precondition

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

Note

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

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>mp</th>
<th>pointer to a memory_pool_t structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>objp</td>
<td>the pointer to the object to be added</td>
</tr>
</tbody>
</table>

Function Class:

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

Here is the call graph for this function:

chPoolAddI → chPoolFreeI → chDbgCheckClassI

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

Initializes an empty guarded memory pool.

Parameters

<table>
<thead>
<tr>
<th>out</th>
<th>gmp</th>
<th>pointer to a guarded_memory_pool_t structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>size</td>
<td>the size of the objects contained in this guarded memory pool, the minimum accepted size is the size of a pointer to void.</td>
</tr>
</tbody>
</table>

Function Class:

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.
Here is the call graph for this function:

![Call Graph](image)

### 7.40.3.16 static cnt_t chGuardedPoolGetCounterI ( guarded_memory_pool_t ∗ gmp ) [inline],[static]

Gets the count of objects in a guarded memory pool.

**Precondition**

The guarded memory pool must be already been initialized.

**Parameters**

- **in gmp** pointer to a `guarded_memory_pool_t` structure

**Returns**

The number of objects.

**Function Class:**

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

Here is the call graph for this function:

![Call Graph](image)

### 7.40.3.17 static void chGuardedPoolAllocI ( guarded_memory_pool_t ∗ gmp ) [inline],[static]

Allocates an object from a guarded memory pool.

**Precondition**

The guarded memory pool must be already been initialized.
Parameters

in gmp pointer to a guarded_memory_pool_t structure

Returns

The pointer to the allocated object.

Return values

NULL if the pool is empty.

Function Class:

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

Here is the call graph for this function:

```
chGuardedPoolFreeI
```

7.40.3.18  static void chGuardedPoolFreeI ( guarded_memory_pool_t * gmp, void * objp ) [inline],[static]

Releases an object into a guarded memory pool.

Precondition

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

Parameters

in gmp pointer to a guarded_memory_pool_t structure
in objp the pointer to the object to be released

Function Class:

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

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

Releases an object into a guarded memory pool.

**Precondition**

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

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>gmp</th>
<th>pointer to a guarded_memory_pool_t structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>objp</td>
<td>the pointer to the object to be released</td>
</tr>
</tbody>
</table>

**Function Class:**

This is an **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:
7.40.3.20 static void chGuardedPoolAdd (guarded_memory_pool_t *gmp, void *objp) [inline],[static]

Adds an object to a guarded memory pool.

Precondition

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

Note

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

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gmp</td>
<td>pointer to a guarded_memory_pool_t structure</td>
</tr>
<tr>
<td>objp</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:

```
chGuardedPoolAdd  
|           |
|           |  
|           |   chGuardedPoolFree |
|           |  
|           |   chSchRescheduleS  |
|           |  
|           |   chSysUnlock       |
|           |  
|           | chGuardedPoolFreeI  |
|           |  
|           |   chSysLock         |
```

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

Adds an object to a guarded memory pool.

Precondition

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

Note

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

<table>
<thead>
<tr>
<th>in</th>
<th>gmp</th>
<th>pointer to a guarded_memory_pool_t structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>objp</td>
<td>the pointer to the object to be added</td>
</tr>
</tbody>
</table>

Function Class:

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

Here is the call graph for this function:

```
chGuardedPoolAddI
    chGuardedPoolFreeI
        chPoolFreeI
            chSemSignalI
```

---

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

Adds an object to a guarded memory pool.

Precondition

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

Note

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

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>gmp</th>
<th>pointer to a guarded_memory_pool_t structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>objp</td>
<td>the pointer to the object to be added</td>
</tr>
</tbody>
</table>

Function Class:

This is an 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 Complex Services

7.41.1 Detailed Description

Modules

- Objects FIFOs
- Dynamic Objects Factory
7.42 Objects FIFOs

7.42.1 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.
7.42 Objects FIFOs

7.42.2 Typedef Documentation

7.42.2.1 typedef struct ch_objects_fifo objects_fifo_t

Type of an objects FIFO.

7.42.3 Function Documentation

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

Initializes a FIFO object.

Precondition

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

Parameters

<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:

```
chFifoObjectInitAligned
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>chGuardedPoolObjectInitAligned</td>
</tr>
<tr>
<td>chPoolObjectInitAligned</td>
</tr>
<tr>
<td>chFifoObjectInit</td>
</tr>
<tr>
<td>chGuardedPoolLoadArray</td>
</tr>
<tr>
<td>chMBObjectInit</td>
</tr>
<tr>
<td>chThdQueueObjectInit</td>
</tr>
<tr>
<td>chGuardedPoolAdd</td>
</tr>
<tr>
<td>chSemObjectInit</td>
</tr>
</tbody>
</table>
```

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

Initializes a FIFO object.
Precondition

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

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>out ofp</code></td>
<td>pointer to a <code>objects_fifo_t</code> structure</td>
</tr>
<tr>
<td><code>in objsize</code></td>
<td>size of objects</td>
</tr>
<tr>
<td><code>in objn</code></td>
<td>number of objects available</td>
</tr>
<tr>
<td><code>in objbuf</code></td>
<td>pointer to the buffer of objects, it must be able to hold <code>objn</code> objects of <code>objsize</code> size with <code>objealign</code> alignment</td>
</tr>
<tr>
<td><code>in msgbuf</code></td>
<td>pointer to the buffer of messages, it must be able to hold <code>objn</code> messages</td>
</tr>
</tbody>
</table>

Function Class:

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

Here is the call graph for this function:

```
chFifoObjectInit    chFifoObjectInitAligned
                   |                        |                    |
                   |                        | chGuardedPoolObjectInitAligned |
                   |                        |                             |
                   chGuardedPoolLoadArray |
                   |                        |
                   | chMBObjectInit |
```

7.42.3.3  static void ∗ chFifoTakeObject( objects_fifo_t * ofp )  [inline],[static]

Allocates a free object.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>in ofp</code></td>
<td>pointer to a <code>objects_fifo_t</code> structure</td>
</tr>
</tbody>
</table>

Returns

The pointer to the allocated object.

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>NULL</code></td>
<td>if an object is not immediately available.</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:

```
chFifoTakeObjectI
  \rightarrow
chGuardedPoolAllocI
  \rightarrow
chPoolAllocI
  \rightarrow
chSemFastWaitI
  \rightarrow
chSemGetCounterI
```

7.42.3.4 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
  \rightarrow
chGuardedPoolAllocTimeoutS
  \rightarrow
chPoolAlloc
  \rightarrow
chSemWaitTimeoutS
```

ChibiOS/RT
7.42.3.5  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:

```
chFifoTakeObjectTimeout → chGuardedPoolAllocTimeout → chSysLock
```

7.42.3.6  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:

```
chFifoReturnObjectI chGuardedPoolFreeI
  - chPoolFreeI
  - chSemSignalI
```

### 7.42.3.7 static void chFifoReturnObjectS ( objects_fifo_t ∗ ofp, void ∗ objp ) [inline],[static]

Releases a fetched object.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>ofp</th>
<th>pointer to a objects_fifo_t structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>objp</td>
<td>pointer to the object to be released</td>
</tr>
</tbody>
</table>

**Function Class:**

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

Here is the call graph for this function:

```
chFifoReturnObjectS chGuardedPoolFreeS
  - chGuardedPoolFreeI
  - chSchRescheduleS
```

### 7.42.3.8 static void chFifoReturnObject ( objects_fifo_t ∗ ofp, void ∗ objp ) [inline],[static]

Releases a fetched object.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>ofp</th>
<th>pointer to a objects_fifo_t structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>objp</td>
<td>pointer to the object to be released</td>
</tr>
</tbody>
</table>

**Function Class:**

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

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

Posts an object.

Note
By design the object can be always immediately posted.

Parameters

| in | ofp | pointer to a objects_fifo_t structure |
| in | objp | pointer to the object to be posted |

Function Class:

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

Here is the call graph for this function:

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

Posts an object.
7.42 Objects FIFOs

**Note**

By design the object can be always immediately posted.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>ofp</th>
<th>pointer to a <code>objects_fifo_t</code> structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>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](call-graph.png)

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

Posts an object.

**Note**

By design the object can be always immediately posted.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>ofp</th>
<th>pointer to a <code>objects_fifo_t</code> structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>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:

```
chFifoSendObject
    |   |
    v   v
chMBPostTimeout
    |   |
    v   v
chMBPostTimeoutS
    |   |
    v   v
chSysUnlock
```

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

Posts an high priority object.

Note

By design the object can be always immediately posted.

Parameters

| in | ofp | pointer to a objects_fifo_t structure |
| in | objp | pointer to the object to be posted |

Function Class:

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

Here is the call graph for this function:

```
chFifoSendObjectAheadI
    |   |
    v   v
chMBPostAhead
    |   |
    v   v
chMBGetFreeCountI
    |   |
    v   v
chThdDequeueNextI
```

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

Posts an high priority object.

Note

By design the object can be always immediately posted.
Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>ofp</th>
<th>pointer to a objects_fifo_t structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>objp</td>
<td>pointer to the object to be posted</td>
</tr>
</tbody>
</table>

Function Class:

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

Here is the call graph for this function:

```
chFifoSendObjectAheadS
chMBPostAheadTimeoutS
chDbgCheckClassS
chMBGetFreeCountI
chThdDequeueNextI
chSchRescheduleS
chThdEnqueueTimeoutS
```

### 7.42.3.14 static void chFifoSendObjectAhead ( objects_fifo_t * ofp, void * objp ) [inline],[static]

Posts an high priority object.

**Note**

By design the object can be always immediately posted.

Parameters

<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:

```
chFifoSendObjectAheadS
chMBPostAheadTimeoutS
chSystLock
```

ChibiOS/RT
7.42.3.15 static msg_t chFifoReceiveObjectI ( objects_fifo_t *ofp, void **objpp ) [inline],[static]

Fetched an object.

Parameters

| in | ofp | pointer to a objects_fifo_t structure |
| in | objpp | pointer to the fetched object reference |

Returns

The operation status.

Return values

| MSG_OK | if an object has been correctly fetched. |
| MSG_TIMEOUT | if the FIFO is empty and a message cannot be fetched. |

Function Class:

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

Here is the call graph for this function:

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

Fetched an object.

Parameters

| in | ofp | pointer to a objects_fifo_t structure |
| in | objpp | pointer to the fetched object reference |
| in | timeout | the number of ticks before the operation timeouts, the following special values are allowed:
| | • TIME_IMMEDIATE immediate timeout. |
| | • TIME_INFINITE no timeout. |
Returns
The operation status.

Return values

| MSG_OK    | if an object has been correctly fetched. |
| MSG_TIMEOUT | if the operation has timed out. |

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

Here is the call graph for this function:

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

Fetches an object.

Parameters

| in    | ofp | pointer to a objects_fifo_t structure |
| in    | objpp | pointer to the fetched object reference |
| in    | timeout | the number of ticks before the operation timeouts, the following special values are allowed: |
|       |       | • TIME_IMMEDIATE immediate timeout. |
|       |       | • TIME_INFINITE no timeout. |

Returns
The operation status.

Return values

| MSG_OK    | if an object has been correctly fetched. |
| 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:
7.43 Dynamic Objects Factory

7.43.1 Detailed Description

The object factory is a subsystem that allows to:

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

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

Precondition

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

Note

Compatible with RT and NIL.

Macros

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

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

### Data Structures

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

### Functions

- void _factory_init (void)
  Initializes the objects factory.
- registered_object_t * chFactoryRegisterObject (const char *name, void *objp)
  Registers a generic object.
- registered_object_t * chFactoryFindObject (const char *name)
  Retrieves a registered object.
7.43 Dynamic Objects Factory

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

- **objects_factory_t ch_factory**
  
  Factory object static instance.

7.43.2  Macro Definition Documentation

7.43.2.1  `#define CH_CFG_FACTORY_MAX_NAMES_LENGTH 8`

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

7.43.2.2  `#define CH_CFG_FACTORY_OBJECTS_REGISTRY TRUE`

Enables the registry of generic objects.

7.43.2.3  `#define CH_CFG_FACTORY_GENERIC_BUFFERS TRUE`

Enables factory for generic buffers.

7.43.2.4  `#define CH_CFG_FACTORY_SEMAPHORES TRUE`

Enables factory for semaphores.

7.43.2.5  `#define CH_CFG_FACTORY_SEMAPHORES FALSE`

Enables factory for semaphores.

7.43.2.6  `#define CH_CFG_FACTORY_MAILBOXES TRUE`

Enables factory for mailboxes.

7.43.2.7  `#define CH_CFG_FACTORY_MAILBOXES FALSE`

Enables factory for mailboxes.

7.43.2.8  `#define CH_CFG_FACTORY_OBJ_FIFO TRUE`

Enables factory for objects FIFOs.

7.43.2.9  `#define CH_CFG_FACTORY_OBJ_FIFO FALSE`

Enables factory for objects FIFOs.

7.43.2.10  `#define CH_CFG_FACTORY_OBJ_FIFO FALSE`

Enables factory for objects FIFOs.
7.43 Dynamic Objects Factory

7.43.2.11 #define CH_CFG_FACTORY_PIPES TRUE

Enables factory for Pipes.

7.43.2.12 #define CH_CFG_FACTORY_PIPES FALSE

Enables factory for Pipes.

7.43.3 Typedef Documentation

7.43.3.1 typedef struct ch_dyn_element dyn_element_t

Type of a dynamic object list element.

7.43.3.2 typedef struct ch_dyn_list dyn_list_t

Type of a dynamic object list.

7.43.3.3 typedef struct ch_registered_static_object registered_object_t

Type of a registered object.

7.43.3.4 typedef struct ch_dyn_object dyn_buffer_t

Type of a dynamic buffer object.

7.43.3.5 typedef struct ch_dyn_semaphore dyn_semaphore_t

Type of a dynamic semaphore.

7.43.3.6 typedef struct ch_dyn_mailbox dyn_mailbox_t

Type of a dynamic buffer object.

7.43.3.7 typedef struct ch_dyn_objects_fifo dyn_objects_fifo_t

Type of a dynamic buffer object.

7.43.3.8 typedef struct ch_dyn_pipe dyn_pipe_t

Type of a dynamic pipe object.

7.43.3.9 typedef struct ch_objects_factory objects_factory_t

Type of the factory main object.
7.43.4 Function Documentation

7.43.4.1 void _factory_init ( void )

Initializes the objects factory.

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

Here is the call graph for this function:

```
    _factory_init
        |          |
        v          v
    chMtxObjectInit
        |          |
        v          v
    chSemObjectInit
        |          |
        v          v
    chPoolObjectInit
        |          |
        v          v
    chPoolObjectInitAligned
        |          |
        v          v
    chCoreAllocAligned
        |          |
        v          v
    chCoreAllocAlignedWithOffset
```

7.43.4.2 registered_object_t * chFactoryRegisterObject ( const char * name, void * objp )

Registers a generic object.

Postcondition
   A reference to the registered object is returned and the reference counter is initialized to one.

Parameters
   
   **name**  name to be assigned to the registered object
   **objp**  pointer to the object to be registered

Returns
   The reference to the registered object.

Return values
   
<table>
<thead>
<tr>
<th>NULL</th>
</tr>
</thead>
</table>
   | if the object to be registered cannot be allocated or a registered object with the same name exists.

Function Class:
   
   Normal API, this function can be invoked by regular system threads but not from within a lock zone.
7.43.4.3  

registered_object_t  * chFactoryFindObject ( const char * name )

Retrieves a registered object.

Postcondition

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

Parameters

| in  | name   | name of the registered object |

Returns

The reference to the found registered object.

Return values

| NULL | if a registered object with the specified name does not exist. |

Function Class:

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

7.43.4.4  

registered_object_t  * chFactoryFindObjectByPointer ( void * objp )

Retrieves a registered object by pointer.

Postcondition

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

Parameters

| in  | 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.
7.43.4.5  void chFactoryReleaseObject ( registered_object_t * rop )

Releases a registered object.
The reference counter of the registered object is decreased by one, if reaches zero then the registered object
memory is freed.

Note
The object itself is not freed, it could be static, only the allocated list element is freed.

Parameters

| in  | rop   | registered object reference |

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

7.43.4.6  dyn_buffer_t * chFactoryCreateBuffer ( const char * name, size_t size )

Creates a generic dynamic buffer object.

Postcondition
A reference to the dynamic buffer object is returned and the reference counter is initialized to one.
The dynamic buffer object is filled with zeros.

Parameters

| in     | name       | name to be assigned to the new dynamic buffer object |
| in     | 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.

7.43.4.7  dyn_buffer_t * chFactoryFindBuffer ( const char * name )

Retrieves a dynamic buffer object.

Postcondition
A reference to the dynamic buffer object is returned with the reference counter increased by one.
Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>name</th>
<th>name of the dynamic buffer object</th>
</tr>
</thead>
</table>

Returns

The reference to the found dynamic buffer object.

Return values

| NULL | if a dynamic buffer object with the specified name does not exist. |

Function Class:

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

7.43.4.8 void chFactoryReleaseBuffer ( dyn_buffer_t * dbp )

Releases a dynamic buffer object.

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

Parameters

<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.

7.43.4.9 dyn_semaphore_t * chFactoryCreateSemaphore ( const char * name, cnt_t n )

Creates a dynamic semaphore object.

Postcondition

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

Parameters

<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.

Here is the call graph for this function:

```
chFactoryCreateSemaphore → chSemObjectInit → queue_init
```

7.43.4.10  

```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.

7.43.4.11  

```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.
7.43 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.

7.43.4.12 dyn_mailbox_t * chFactoryCreateMailbox ( const char * name, size_t n )

Creates a dynamic mailbox object.

Postcondition

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

Parameters

| in   | 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.

Here is the call graph for this function:

7.43.4.13 dyn_mailbox_t * chFactoryFindMailbox ( const char * name )

Retrieves a dynamic mailbox object.

Postcondition

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

| in | 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.

7.43.4.14 void chFactoryReleaseMailbox ( dyn_mailbox_t * dmp )

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

Parameters

| in | 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.

7.43.4.15 dyn_objects_fifo_t * chFactoryCreateObjectsFIFO ( const char * name, size_t objsize, size_t objn, unsigned objalign )

Creates a dynamic "objects FIFO" object.

Postcondition

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

Parameters

| in | name | name to be assigned to the new dynamic "objects FIFO" object |
| in | objsize | size of objects |
| in | objn | number of objects available |
| in | objalign | required objects alignment |
Returns

The reference to the created dynamic "objects FIFO" object.

Return values

| NULL | if the dynamic "objects FIFO" object cannot be allocated or a dynamic "objects FIFO" object with the same name exists. |

Function Class:

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

Here is the call graph for this function:

7.43.4.16 \texttt{dyn\_objects\_fifo\_t = chFactoryFindObject\_FIFO ( 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.
7.43.4.17 void chFactoryReleaseObjectsFIFO ( dyn_objects_fifo_t *dofp )

Releases a dynamic "objects FIFO" object.
The reference counter of the dynamic "objects FIFO" object is decreased by one, if reaches zero then the dynamic "objects FIFO" object memory is freed.

Parameters

| in  | dof | dynamic "objects FIFO" object reference |

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

7.43.4.18 dyn_pipe_t * chFactoryCreatePipe ( const char *name, size_t size )

Creates a dynamic pipe object.

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

Parameters

| in  | 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.

Here is the call graph for this function:

```
   chFactoryCreatePipe -> chPipeObjectInit
```
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.

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.

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.

7.43.4.22 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.

7.43.4.23 static size_t chFactoryGetBufferSize ( dyn_buffer_t * dbp ) [inline],[static]

Returns the size of a generic dynamic buffer object.

Parameters

| in | dbp | dynamic buffer object reference |

Returns

The size of the buffer object in bytes.

Function Class:

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

Here is the call graph for this function:

![Call Graph]

7.43.4.24 static uint8_t* chFactoryGetBuffer ( dyn_buffer_t * dbp ) [inline],[static]

Returns the pointer to the inner buffer.
7.43 Dynamic Objects Factory

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.

7.43.4.25 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.

7.43.4.26 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.

7.43.4.27 static objects_fifo_t* chFactoryGetObjectsFIFO ( dyn_objects_fifo_t* dof ) [inline],[static]

Returns the pointer to the inner objects FIFO.

Parameters

| in   | dof  | dynamic "objects FIFO" object reference |

ChibiOS/RT
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.

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

Returns the pointer to the inner pipe.

Parameters

| in | dpp | dynamic pipe object reference |

Returns

The pointer to the pipe.

Function Class:

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

7.43.5 Variable Documentation

7.43.5.1 objects_factory_t ch_factory

Factory object static instance.

Note

It is a global object because it could be accessed through a specific debugger plugin.
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_dyn_element Struct Reference

Type of a dynamic object list element.
#include <chfactory.h>
Collaboration diagram for ch_dyn_element:

```
ch_dyn_element
+ refs
+ name
+next
```

Data Fields

- struct `ch_dyn_element` * next
  
  *Next dynamic object in the list.*

- ucnt_t `refs`

  *Number of references to this object.*

8.2.1 Detailed Description

Type of a dynamic object list element.

8.2.2 Field Documentation

8.2.2.1 struct `ch_dyn_element` * `ch_dyn_element`::next

Next dynamic object in the list.

8.2.2.2 ucnt_t `ch_dyn_element`::refs

Number of references to this object.

8.3 ch_dyn_list Struct Reference

Type of a dynamic object list.

```
#include <chfactory.h>
```
Collaboration diagram for ch_dyn_list:

8.3.1 Detailed Description

Type of a dynamic object list.

8.4 ch_dyn_mailbox Struct Reference

Type of a dynamic buffer object.
#include <chfactory.h>
Collaboration diagram for ch_dyn_mailbox:

Data Fields

- **dyn_element_t element**
  
  List element of the dynamic buffer object.

- **mailbox_t mbx**
  
  The mailbox.

- **msg_t msgbuf []**
  
  Messages buffer.

8.4.1 Detailed Description

Type of a dynamic buffer object.

8.4.2 Field Documentation

8.4.2.1 **dyn_element_t ch_dyn_mailbox::element**

List element of the dynamic buffer object.
8.4.2.2 mailbox_t ch_dyn_mailbox::mbx

The mailbox.

8.4.2.3 msg_t ch_dyn_mailbox::msgbuf[]

Messages buffer.

Note

This requires C99.

8.5 ch_dyn_object Struct Reference

Type of a dynamic buffer object.

#include <chfactory.h>

Collaboration diagram for ch_dyn_object:

Data Fields

- **dyn_element_t element**
  
  *List element of the dynamic buffer object.*

- **uint8_t buffer[]**
  
  *The buffer.*

8.5.1 Detailed Description

Type of a dynamic buffer object.
8.5.2 Field Documentation

8.5.2.1 dyn_element_t ch_dyn_object::element

List element of the dynamic buffer object.

8.5.2.2 uint8_t ch_dyn_object::buffer[]

The buffer.

Note
This requires C99.

8.6 ch_dyn_objects_fifo Struct Reference

Type of a dynamic buffer object.
#include <chfactory.h>

Collaboration diagram for ch_dyn_objects_fifo:
Data Fields

- **dyn_element_t element**
  
  *List element of the dynamic buffer object.*

- **objects_fifo_t fifo**
  
  *The objects FIFO.*

- **msg_t msgbuf []**
  
  *Messages buffer.*

### 8.6.1 Detailed Description

Type of a dynamic buffer object.

### 8.6.2 Field Documentation

#### 8.6.2.1 dyn_element_t ch_dyn_objects_fifo::element

List element of the dynamic buffer object.

#### 8.6.2.2 objects_fifo_t ch_dyn_objects_fifo::fifo

The objects FIFO.

#### 8.6.2.3 msg_t ch_dyn_objects_fifo::msgbuf[]

Messages buffer.
Note

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

8.7 ch_dyn_pipe Struct Reference

Type of a dynamic pipe object.
#include <chfactory.h>
Collaboration diagram for ch_dyna_pipe:

Data Fields

- **dyn_element_t element**
  
  List element of the dynamic pipe object.

- **pipe_t pipe**
  
  The pipe.

- **uint8_t buffer []**
8.7.1 Detailed Description

Type of a dynamic pipe object.

8.7.2 Field Documentation

8.7.2.1 dyn_element_t ch_dyn_pipe::element

List element of the dynamic pipe object.

8.7.2.2 pipe_t ch_dyn_pipe::pipe

The pipe.

8.7.2.3 uint8_t ch_dyn_pipe::buffer[]

Messages buffer.

Note

This requires C99.

8.8 ch_dyn_semaphore Struct Reference

Type of a dynamic semaphore.

#include <chfactory.h>
Collaboration diagram for ch_dyn_semaphore:

```
Data Fields

- dyn_element_t element
  List element of the dynamic semaphore.
- semaphore_t sem
  The semaphore.

8.8.1 Detailed Description

Type of a dynamic semaphore.

8.8.2 Field Documentation

8.8.2.1 dyn_element_t ch_dyn_semaphore::element

List element of the dynamic semaphore.

8.8.2.2 semaphore_t ch_dyn_semaphore::sem

The semaphore.
8.9 ch_mutex Struct Reference

Mutex structure.

#include <chmtx.h>

Collaboration diagram for ch_mutex:
Data Fields

- **threads_queue_t queue**
  
  Queue of the threads sleeping on this mutex.

- **thread_t * owner**
  
  Owner thread_t pointer or NULL.

- **mutex_t * next**
  
  Next mutex_t into an owner-list or NULL.

- **cnt_t cnt**
  
  Mutex recursion counter.

8.9.1 Detailed Description

Mutex structure.

8.9.2 Field Documentation

8.9.2.1 **threads_queue_t ch_mutex::queue**

Queue of the threads sleeping on this mutex.

8.9.2.2 **thread_t * ch_mutex::owner**

Owner thread_t pointer or NULL.

8.9.2.3 **mutex_t * ch_mutex::next**

Next mutex_t into an owner-list or NULL.

8.9.2.4 **cnt_t ch_mutex::cnt**

Mutex recursion counter.

8.10 **ch_objects_factory Struct Reference**

Type of the factory main object.

#include <chfactory.h>
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**
8.10.1 Detailed Description

Type of the factory main object.

8.10.2 Field Documentation

8.10.2.1 mutex_t ch_objects_factory::mtx

Factory access mutex or semaphore.

8.10.2.2 dyn_list_t ch_objects_factory::obj_list

List of the registered objects.

8.10.2.3 memory_pool_t ch_objects_factory::obj_pool

Pool of the available registered objects.

8.10.2.4 dyn_list_t ch_objects_factory::buf_list

List of the allocated buffer objects.

8.10.2.5 dyn_list_t ch_objects_factory::sem_list

List of the allocated semaphores.

8.10.2.6 memory_pool_t ch_objects_factory::sem_pool

Pool of the available semaphores.

8.10.2.7 dyn_list_t ch_objects_factory::mbx_list

List of the allocated buffer objects.

8.10.2.8 dyn_list_t ch_objects_factory::fifo_list

List of the allocated "objects FIFO" objects.

8.10.2.9 dyn_list_t ch_objects_factory::pipe_list

List of the allocated pipe objects.
8.11 ch_objects_fifo Struct Reference

Type of an objects FIFO.

```c
#include <chobjfifos.h>
```

Collaboration diagram for ch_objects_fifo:

```
ch_threads_queue

+mq
+qw
+queue

memory_pool_t

+object_size
+align
+provider

ch_semaphore

+cnt

mailbox_t

+buffer
+top
+wrptr
+rdptr
+cnt
+reset

guarded_memory_pool_t

+free
+sem
+pool

ch_objects_fifo

+mbx
```

Data Fields

- `guarded_memory_pool_t` free
Pool of the free objects.

- mailbox_t mbx
  Mailbox of the sent objects.

### 8.11.1 Detailed Description

Type of an objects FIFO.

### 8.11.2 Field Documentation

#### 8.11.2.1 guarded_memory_pool_t ch_objects_fifo::free

Pool of the free objects.

#### 8.11.2.2 mailbox_t ch_objects_fifo::mbx

Mailbox of the sent objects.

### 8.12 ch_registered_static_object Struct Reference

Type of a registered object.

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

Collaboration diagram for ch_registered_static_object:

![Collaboration Diagram](image)

**Data Fields**

- dyn_element_t element
  
  *List element of the registered object.*
- void *objp
8.12.1 Detailed Description

Type of a registered object.

8.12.2 Field Documentation

8.12.2.1 `dyn_element_t` ch_registered_static_object::element

List element of the registered object.

8.12.2.2 `void*` ch_registered_static_object::objp

Pointer to the object.

Note

The type of the object is not stored in anyway.

8.13 ch_semaphore Struct Reference

Semaphore structure.

```cpp
#include <chsem.h>
```

Inheritance diagram for ch_semaphore:
Collaboration diagram for `ch_semaphore`:

Data Fields

- `threads_queue_t queue`
  Queue of the threads sleeping on this semaphore.

- `cnt_t cnt`
  The semaphore counter.

8.13.1 Detailed Description

Semaphore structure.

8.13.2 Field Documentation

8.13.2.1 `threads_queue_t ch_semaphore::queue`

Queue of the threads sleeping on this semaphore.
8.14 ch_system Struct Reference

System data structure.
#include <chschd.h>

Collaboration diagram for ch_system:

Data Fields

- ready_list_t rlist
  Ready list header.
- virtual_timers_list_t vtlist
  Virtual timers delta list header.
- system_debug_t dbg
  System debug.
- thread_t mainthread
  Main thread descriptor.
- tm_calibration_t tm
  Time measurement calibration data.
- kernel_stats_t kernel_stats
  Global kernel statistics.

8.14.1 Detailed Description

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

8.14.2 Field Documentation

8.14.2.1 ready_list_t ch_system::rlist

Ready list header.

8.14.2.2 virtual_timers_list_t ch_system::vtlist

Virtual timers delta list header.

8.14.2.3 system_debug_t ch_system::dbg

System debug.

8.14.2.4 thread_t ch_system::mainthread

Main thread descriptor.

8.14.2.5 tm_calibration_t ch_system::tm

Time measurement calibration data.

8.14.2.6 kernel_stats_t ch_system::kernel_stats

Global kernel statistics.

8.15 ch_system_debug Struct Reference

System debug data structure.
#include <chschd.h>
Collaboration diagram for `ch_system_debug`:

![Collaboration Diagram](image)

**Data Fields**

- **const char *volatile panic_msg**
  
  *Pointer to the panic message.*

- **cnt_t isr_cnt**
  
  *ISR nesting level.*

- **cnt_t lock_cnt**
  
  *Lock nesting level.*

- **ch_trace_buffer_t trace_buffer**
  
  *Public trace buffer.*
8.15.1 Detailed Description

System debug data structure.

8.15.2 Field Documentation

8.15.2.1 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.

8.15.2.2 cnt_t ch_system_debug::isr_cnt

ISR nesting level.

8.15.2.3 cnt_t ch_system_debug::lock_cnt

Lock nesting level.

8.15.2.4 ch_trace_buffer_t ch_system_debug::trace_buffer

Public trace buffer.

8.16 ch_thread Struct Reference

Structure representing a thread.
#include <chschd.h>
Collaboration diagram for ch_thread:

Data Fields

- `threads_queue_t queue`
  Threads queue header.
- `tprio_t prio`
  Thread priority.
- `struct port_context ctx`
  Processor context.
- `thread_t * newer`
  Newer registry element.
- `thread_t * older`
  Older registry element.
- `const char * name`
  Thread name or `NULL`.
- `stkalign_t * wabase`
  Working area base address.
- `tstate_t * state`
  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.

```c
union {
  msg_t rdymsg
      Thread wakeup code.
  msg_t exitcode
      Thread exit code.
  void * wtpobjp
      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 * wtmtp
      Pointer to a generic mutex object.
  eventmask_t ewmask
      Enabled events mask.
} u
```

State-specific fields.

- `threads_list_t * waiting`
  Termination waiting list.
- `threads_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.
8.16.1 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.

8.16.2 Field Documentation

8.16.2.1 threads_queue_t ch_thread::queue

Threads queue header.

8.16.2.2 tprio_t ch_thread::prio

Thread priority.

8.16.2.3 struct port_context ch_thread::ctx

Processor context.

8.16.2.4 thread_t* ch_thread::newer

Newer registry element.

8.16.2.5 thread_t* ch_thread::older

Older registry element.

8.16.2.6 const char* ch_thread::name

Thread name or NULL.

8.16.2.7 stkalign_t* ch_thread::wabase

Working area base address.

Note

This pointer is used for stack overflow checks and for dynamic threading.

8.16.2.8 tstate_t ch_thread::state

Current thread state.

8.16.2.9 tmode_t ch_thread::flags

Various thread flags.
8.16.2.10 trefs_t ch_thread::refs

References to this thread.

8.16.2.11 tslices_t ch_thread::ticks

Number of ticks remaining to this thread.

8.16.2.12 volatile systime_t ch_thread::time

Thread consumed time in ticks.

Note

This field can overflow.

8.16.2.13 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.

8.16.2.14 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 chThd←Wait() on this thread.

8.16.2.15 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.

8.16.2.16 thread_reference_t* ch_thread::wttrp

Pointer to a generic thread reference object.

Note

This field is used to get a pointer to a synchronization object and is valid when the thread is in CH_STATE←_SUSPENDED state.
8.16.2.17  msg_t ch_thread::sentmsg

Thread sent message.

8.16.2.18  struct ch_semaphore* ch_thread::wtsemp

Pointer to a generic semaphore object.

Note
This field is used to get a pointer to a synchronization object and is valid when the thread is in CH_STATE←_WTSEM state.

8.16.2.19  struct ch_mutex* ch_thread::wtmxp

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.

8.16.2.20  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.

8.16.2.21  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.

8.16.2.22  threads_list_t ch_thread::waiting

Termination waiting list.

8.16.2.23  threads_queue_t ch_thread::msgqueue

Messages queue.

8.16.2.24  eventmask_t ch_thread::epending

Pending events mask.
8.16.2.25 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.

8.16.2.26 tprio_t ch_thread::realprio

Thread's own, non-inherited, priority.

8.16.2.27 void * ch_thread::mpool

Memory Pool where the thread workspace is returned.

8.16.2.28 time_measurement_t ch_thread::stats

Thread statistics.

8.17 ch_threads_list Struct Reference

Generic threads single link list, it works like a stack.

```c
#include <chschd.h>
```
Data Fields

- `thread_t` * next

   *Next in the list/queue.*
8.17.1 Detailed Description

Generic threads single link list, it works like a stack.

8.17.2 Field Documentation

8.17.2.1 thread_t * ch_threads_list::next

Next in the list/queue.

8.18 ch_threads_queue Struct Reference

Generic threads bidirectional linked list header and element.

#include <chschd.h>

Inherited by ch_ready_list.

Collaboration diagram for ch_threads_queue:
8.19 ch_trace_buffer_t Struct Reference

Trace buffer header.

#include <chtrace.h>
Data Fields

- **uint16_t suspended**
  
  *Suspended trace sources mask.

- **uint16_t size**
  
  *Trace buffer size (entries).*

- **ch_trace_event_t * ptr**
8.20 ch_trace_event_t Struct Reference

Trace buffer record.

#include <chtrace.h>

ChibiOS/RT
Collaboration diagram for ch_trace_event_t:

Data Fields

- **uint32_t type**:3
  
  Record type.

- **uint32_t state**:5
  
  Switched out thread state.

- **uint32_t rtstamp**:24
  
  Accurate time stamp.

- **systime_t time**
  
  System time stamp of the switch event.

- **thread_t * ntp**
  
  Switched in thread.

- **void * wtobjp**
  
  Object where going to sleep.
8.20 ch_trace_event_t Struct Reference

- struct {
  thread_t * ntp
  Switched in thread.
  void * wtobjp
  Object where going to sleep.
  }

sw

Structure representing a context switch.

- const char * name
  ISR function name taken using func.

isr

Structure representing an ISR enter.

- const char * reason
  Halt error string.

halt

Structure representing an halt.

- void * up1
  Trace user parameter 1.

- void * up2
  Trace user parameter 2.

user

User trace structure.

8.20.1 Detailed Description

Trace buffer record.

8.20.2 Field Documentation

8.20.2.1 uint32_t ch_trace_event_t::type

Record type.

8.20.2.2 uint32_t ch_trace_event_t::state

Switched out thread state.
8.20.2.3 uint32_t ch_trace_event_t::rtstamp

Accurate time stamp.

Note

This field only available if the post supports PORT_SUPPORTS_RT else it is set to zero.

8.20.2.4 systime_t ch_trace_event_t::time

System time stamp of the switch event.

8.20.2.5 thread_t* ch_trace_event_t::ntp

Switched in thread.

8.20.2.6 void* ch_trace_event_t::wtobjp

Object where going to sleep.

8.20.2.7 struct {...} ch_trace_event_t::sw

Structure representing a context switch.

8.20.2.8 const char* ch_trace_event_t::name

ISR function name taken using func.

8.20.2.9 struct {...} ch_trace_event_t::isr

Structure representing an ISR enter.

8.20.2.10 const char* ch_trace_event_t::reason

Halt error string.

8.20.2.11 struct {...} ch_trace_event_t::halt

Structure representing an halt.

8.20.2.12 void* ch_trace_event_t::up1

Trace user parameter 1.

8.20.2.13 void* ch_trace_event_t::up2

Trace user parameter 2.
8.20.2.14 struct {...} ch_trace_event_t::user

User trace structure.

8.21 ch_virtual_timer Struct Reference

Virtual Timer descriptor structure.

#include <chschd.h>

Inheritance diagram for ch_virtual_timer:
Collaboration diagram for ch_virtual_timer:

Data Fields

- virtual_timer_t ∗ next
  Next timer in the list.
- virtual_timer_t ∗ prev
  Previous timer in the list.
- sysinterval_t delta
  Time delta before timeout.
- vtfunc_t func
  Timer callback function pointer.
- void ∗ par
  Timer callback function parameter.

8.21.1 Detailed Description

Virtual Timer descriptor structure.

8.21.2 Field Documentation

8.21.2.1 virtual_timer_t ∗ ch_virtual_timer::next

Next timer in the list.

8.21.2.2 virtual_timer_t ∗ ch_virtual_timer::prev

Previous timer in the list.
8.21.2.3  `sysinterval_t ch_virtual_timer::delta`

Time delta before timeout.

8.21.2.4  `vtfunc_t ch_virtual_timer::func`

Timer callback function pointer.

8.21.2.5  `void* ch_virtual_timer::par`

Timer callback function parameter.

8.22  `ch_virtual_timers_list Struct Reference`

Virtual timers list header.

```
#include <chschd.h>
```

Inheritance diagram for `ch_virtual_timers_list`:
Collaboration diagram for ch_virtual_timers_list:

Data Fields

- virtual_timer_t * next
  Next timer in the delta list.
- virtual_timer_t * prev
  Last timer in the delta list.
- sysinterval_t delta
  Must be initialized to -1.
- volatile systime_t systime
  System Time counter.
- systime_t lasttime
  System time of the last tick event.

8.22.1 Detailed Description

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.

8.22.2 Field Documentation

8.22.2.1 virtual_timer_t = ch_virtual_timers_list::next

Next timer in the delta list.
8.22.2.2 virtual_timer_t ch_virtual_timers_list::prev

Last timer in the delta list.

8.22.2.3 sysinterval_t ch_virtual_timers_list::delta

Must be initialized to -1.

8.22.2.4 volatile systime_t ch_virtual_timers_list::systime

System Time counter.

8.22.2.5 systime_t ch_virtual_timers_list::lasttime

System time of the last tick event.

8.23 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.
8.23.1 Detailed Description

ChibiOS/RT memory signature record.

8.23.2 Field Documentation

8.23.2.1 char chdebug_t::identifier[4]

Always set to "main".

8.23.2.2 uint8_t chdebug_t::zero

Must be zero.

8.23.2.3 uint8_t chdebug_t::size

Size of this structure.
8.23.2.4  uint16_t chdebug_t::version
Encoded ChibiOS/RT version.

8.23.2.5  uint8_t chdebug_t::ptrsize
Size of a pointer.

8.23.2.6  uint8_t chdebug_t::timesize
Size of a systime_t.

8.23.2.7  uint8_t chdebug_t::threadsize
Size of a thread_t.

8.23.2.8  uint8_t chdebug_t::off_prio
Offset of prio field.

8.23.2.9  uint8_t chdebug_t::off_ctx
Offset of ctx field.

8.23.2.10 uint8_t chdebug_t::off_newer
Offset of newer field.

8.23.2.11 uint8_t chdebug_t::off_older
Offset of older field.

8.23.2.12 uint8_t chdebug_t::off_name
Offset of name field.

8.23.2.13 uint8_t chdebug_t::off_stklimit
Offset of stklimit field.

8.23.2.14 uint8_t chdebug_t::off_state
Offset of state field.

8.23.2.15 uint8_t chdebug_t::off_flags
Offset of flags field.
8.23.2.16 uint8_t chdebug_t::off_refs

Offset of refs field.

8.23.2.17 uint8_t chdebug_t::off_preempt

Offset of preempt field.

8.23.2.18 uint8_t chdebug_t::off_time

Offset of time field.

8.24 condition_variable Struct Reference

condition_variable_t structure.

#include <chcond.h>

Collaboration diagram for condition_variable:

Data Fields

- threads_queue_t queue
  
  Condition variable threads queue.
8.24.1 Detailed Description

condition_variable_t structure.

8.24.2 Field Documentation

8.24.2.1 threads_queue_t condition_variable::queue

Condition variable threads queue.

8.25 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.

8.25.1 Detailed Description

Event Listener structure.

8.25.2 Field Documentation

8.25.2.1 `event_listener_t*` `event_listener::next`

Next Event Listener registered on the event source.

8.25.2.2 `thread_t` `event_listener::listener`

Thread interested in the event source.

8.25.2.3 `eventmask_t` `event_listener::events`

Events to be set in the listening thread.

8.25.2.4 `eventflags_t` `event_listener::flags`

Flags added to the listener by the event source.

8.25.2.5 `eventflags_t` `event_listener::wflags`

Flags that this listener interested in.

8.26 `event_source Struct Reference`

Event Source structure.

#include <chevents.h>
Collaboration diagram for event_source:

Data Fields

- `event_listener_t * next`
  
  First Event Listener registered on the Event Source.

8.26.1 Detailed Description

Event Source structure.

8.26.2 Field Documentation

8.26.2.1 `event_listener_t * event_source::next`

First Event Listener registered on the Event Source.
8.27 guarded_memory_pool_t Struct Reference

Guarded memory pool descriptor.

```
#include <chmempools.h>
```

Collaboration diagram for guarded_memory_pool_t:

Data Fields

- `semaphore_t sem`
  
  Counter semaphore guarding the memory pool.

- `memory_pool_t pool`
  
  The memory pool itself.

8.27.1 Detailed Description

Guarded memory pool descriptor.

8.27.2 Field Documentation

8.27.2.1 `semaphore_t guarded_memory_pool_t::sem`

Counter semaphore guarding the memory pool.
8.27.2.2 memory_pool_t guarded_memory_pool_t::pool

The memory pool itself.

8.28 heap_header Union Reference

Memory heap block header.

#include <chmemheaps.h>

Collaboration diagram for heap_header:

8.28.1 Detailed Description

Memory heap block header.

8.28.2 Field Documentation

8.28.2.1 heap_header_t∗ heap_header::next

Next block in free list.

8.28.2.2 size_t heap_header::pages

Size of the area in pages.

8.28.2.3 memory_heap_t∗ heap_header::heap

Block owner heap.
8.28.2.4 size_t heap_header::size

Size of the area in bytes.

8.29 kernel_stats_t Struct Reference

Type of a kernel statistics structure.

#include <chstats.h>

Collaboration diagram for kernel_stats_t:

![Collaboration Diagram](image)

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.

8.29.1 Detailed Description

Type of a kernel statistics structure.
8.29.2 Field Documentation

8.29.2.1 ucnt_t kernel_stats_t::n_irq

Number of IRQs.

8.29.2.2 ucnt_t kernel_stats_t::n_ctxswc

Number of context switches.

8.29.2.3 time_measurement_t kernel_stats_t::m_crit_thd

Measurement of threads critical zones duration.

8.29.2.4 time_measurement_t kernel_stats_t::m_crit_isr

Measurement of ISRs critical zones duration.

8.30 mailbox_t Struct Reference

Structure representing a mailbox object.

#include <chmboxes.h>

Collaboration diagram for mailbox_t:
Data Fields

- `msg_t * buffer`
  Pointer to the mailbox buffer.
- `msg_t * top`
  Pointer to the location after the buffer.
- `msg_t * wrptr`
  Write pointer.
- `msg_t * rdptr`
  Read pointer.
- `size_t cnt`
  Messages in queue.
- `bool reset`
  True in reset state.
- `threads_queue_t qw`
  Queued writers.
- `threads_queue_t qr`
  Queued readers.

8.30.1 Detailed Description
Structure representing a mailbox object.

8.30.2 Field Documentation

8.30.2.1 `msg_t * mailbox_t::buffer`
Pointer to the mailbox buffer.

8.30.2.2 `msg_t * mailbox_t::top`
Pointer to the location after the buffer.

8.30.2.3 `msg_t * mailbox_t::wrptr`
Write pointer.

8.30.2.4 `msg_t * mailbox_t::rdptr`
Read pointer.

8.30.2.5 `size_t mailbox_t::cnt`
Messages in queue.

8.30.2.6 `bool mailbox_t::reset`
True in reset state.
8.30.2.7 threads_queue_t mailbox_t::qw

Queued writers.

8.30.2.8 threads_queue_t mailbox_t::qr

Queued readers.

8.31 memcore_t Struct Reference

Type of memory core object.
#include <chmemcore.h>

Collaboration diagram for memcore_t:

```
memcore_t
+ nextmem
+ endmem
```

Data Fields

- uint8_t * nextmem
  Next free address.
- uint8_t * endmem
  Final address.

8.31.1 Detailed Description

Type of memory core object.

8.31.2 Field Documentation

8.31.2.1 uint8_t * memcore_t::nextmem

Next free address.

8.31.2.2 uint8_t * memcore_t::endmem

Final address.
8.32 memory_heap Struct Reference

Structure describing a memory heap.

```c
#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.
8.32.1 Detailed Description
Structure describing a memory heap.

8.32.2 Field Documentation

8.32.2.1 memgetfunc2_t memory_heap::provider
Memory blocks provider for this heap.

8.32.2.2 heap_header_t memory_heap::header
Free blocks list header.

8.32.2.3 mutex_t memory_heap::mtx
Heap access mutex.

8.33 memory_pool_t Struct Reference
Memory pool descriptor.
#include <chmempools.h>
Collaboration diagram for memory_pool_t:

```
Data Fields

- struct pool_header * next
  Pointer to the header.
- size_t object_size
```

Memory pool objects size.

- unsigned align
  Required alignment.
- memgetfunc_t provider
  Memory blocks provider for this pool.

8.33.1 Detailed Description

Memory pool descriptor.

8.33.2 Field Documentation

8.33.2.1 struct pool_header* memory_pool_t::next

Pointer to the header.

8.33.2.2 size_t memory_pool_t::object_size

Memory pool objects size.

8.33.2.3 unsigned memory_pool_t::align

Required alignment.

8.33.2.4 memgetfunc_t memory_pool_t::provider

Memory blocks provider for this pool.

8.34 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`

ChibiOS/RT
Write pointer.
• uint8_t ∗ rdptr
  Read pointer.
• size_t cnt
  Bytes in the pipe.
• bool reset
  True if in reset state.
• thread_reference_t wtr
  Waiting writer.
• thread_reference_t rtr
  Waiting reader.
• mutex_t cmtx
  Common access mutex.
• mutex_t wmtx
  Write access mutex.
• mutex_t rmtx
  Read access mutex.

8.34.1 Detailed Description
Structure representing a pipe object.

8.34.2 Field Documentation
8.34.2.1 uint8_t ∗ pipe_t::buffer
Pointer to the pipe buffer.

8.34.2.2 uint8_t ∗ pipe_t::top
Pointer to the location after the buffer.

8.34.2.3 uint8_t ∗ pipe_t::wrptr
Write pointer.

8.34.2.4 uint8_t ∗ pipe_t::rdptr
Read pointer.

8.34.2.5 size_t pipe_t::cnt
Bytes in the pipe.

8.34.2.6 bool pipe_t::reset
True if in reset state.
8.34.2.7 thread_reference_t pipe_t::wtr

Waiting writer.

8.34.2.8 thread_reference_t pipe_t::rtr

Waiting reader.

8.34.2.9 mutex_t pipe_t::cmtx

Common access mutex.

8.34.2.10 mutex_t pipe_t::wmtx

Write access mutex.

8.34.2.11 mutex_t pipe_t::rmtx

Read access mutex.

8.35 pool_header Struct Reference

Memory pool free object header.

#include <chmempools.h>

Collaboration diagram for pool_header:

![Collaboration diagram](image)

Data Fields

- struct pool_header * next

  *Pointer to the next pool header in the list.*

8.35.1 Detailed Description

Memory pool free object header.
8.35.2 Field Documentation

8.35.2.1 struct pool_header+ pool_header::next

Pointer to the next pool header in the list.

8.36 thread_descriptor_t Struct Reference

Type of a thread descriptor.

#include <chthreads.h>

Collaboration diagram for thread_descriptor_t:

```
+ name
+ wbase
+ wend
+ prio
+ funcp
+ arg
```

Data Fields

- const char * name
  
  Thread name.

- stkalign_t * wbase

  Pointer to the working area base.

- stkalign_t * wend

  End of the working area.

- tpio_t prio

  Thread priority.

- tfunc_t funcp

  Thread function pointer.

- void * arg

  Thread argument.

8.36.1 Detailed Description

Type of a thread descriptor.
8.36.2 Field Documentation

8.36.2.1 const char* thread_descriptor_t::name

Thread name.

8.36.2.2 stkalign_t* thread_descriptor_t::wbase

Pointer to the working area base.

8.36.2.3 stkalign_t* thread_descriptor_t::wend

End of the working area.

8.36.2.4 t prio_t thread_descriptor_t::prio

Thread priority.

8.36.2.5 tfunc_t thread_descriptor_t::funcp

Thread function pointer.

8.36.2.6 void* thread_descriptor_t::arg

Thread argument.

8.37 time_measurement_t Struct Reference

Type of a Time Measurement object.

#include <cthm.h>

Collaboration diagram for time_measurement_t:

```
+ best
+ worst
+ last
+ n
+ cumulative
```

Data Fields

- rtcnt_t best
8.37.1 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.

8.37.2 Field Documentation

8.37.2.1 rtcnt_t time_measurement_t::best

Best measurement.

8.37.2.2 rtcnt_t time_measurement_t::worst

Worst measurement.

8.37.2.3 rtcnt_t time_measurement_t::last

Last measurement.

8.37.2.4 ucnt_t time_measurement_t::n

Number of measurements.

8.37.2.5 rttime_t time_measurement_t::cumulative

Cumulative measurement.

8.38 tm_calibration_t Struct Reference

Type of a time measurement calibration data.

#include <chtm.h>
Collaboration diagram for tm_calibration_t:

```
<table>
<thead>
<tr>
<th>tm_calibration_t</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ offset</td>
</tr>
</tbody>
</table>
```

Data Fields

- rtcnt_t offset

  *Measurement calibration value.*

8.38.1 Detailed Description

Type of a time measurement calibration data.

8.38.2 Field Documentation

8.38.2.1 rtcnt_t tm_calibration_t::offset

Measurement calibration value.
Chapter 9

File Documentation

9.1 ch.h File Reference

ChibiOS/RT main include file.

```c
#include "chconf.h"
#include "chchecks.h"
#include "chlicense.h"
#include "chrestrictions.h"
#include "chtypes.h"
#include "chsystypes.h"
#include "chdebug.h"
#include "ctime.h"
#include "chalign.h"
#include "chcore.h"
#include "chtrace.h"
#include "chtm.h"
#include "chstats.h"
#include "chschd.h"
#include "chsystats.h"
#include "chpht.h"
#include "chthreads.h"
#include "chregistry.h"
#include "chsem.h"
#include "chttx.h"
#include "chcond.h"
#include "chevents.h"
#include "chmsg.h"
#include "chlib.h"
#include "chdynamic.h"
```

Macros

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

ChibiOS/RT version identification

- `#define CH_KERNEL_VERSION "6.0.4"`
Kernel version string.

- `#define CH_KERNEL_MAJOR 6`  
  Kernel version major number.
- `#define CH_KERNEL_MINOR 0`  
  Kernel version minor number.
- `#define CH_KERNEL_PATCH 4`  
  Kernel version patch number.

Constants for configuration options

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

Functions

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

9.1.1 Detailed Description

ChibiOS/RT main include file.

9.2 chalign.h File Reference

Memory alignment macros and structures.

Macros

Memory alignment support macros

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

9.2.1 Detailed Description

Memory alignment macros and structures.

9.3 chbsem.h File Reference

Binary semaphores structures and macros.
Data Structures

- struct ch_binary_semaphore
  
  Binary semaphore type.

Macros

- #define _BSEMAPHORE_DATA(name, taken) (_SEMAPHORE_DATA(name.sem, ((taken) ? 0 : 1)))
  
  Data part of a static semaphore initializer.

- #define BSEMAPHORE_DECL(name, taken) binary_semaphore_t name = _BSEMAPHORE_DATA(name, taken)
  
  Static semaphore initializer.

Typedefs

- typedef struct ch_binary_semaphore binary_semaphore_t
  
  Binary semaphore type.

Functions

- static void chBSemObjectInit (binary_semaphore_t *bsp, bool taken)
  
  Initializes a binary semaphore.

- static msg_t chBSemWait (binary_semaphore_t *bsp)
  
  Wait operation on the binary semaphore.

- static msg_t chBSemWaitS (binary_semaphore_t *bsp)
  
  Wait operation on the binary semaphore.

- static msg_t chBSemWaitTimeoutS (binary_semaphore_t *bsp, sysinterval_t timeout)
  
  Wait operation on the binary semaphore.

- static msg_t chBSemWaitTimeout (binary_semaphore_t *bsp, sysinterval_t timeout)
  
  Wait operation on the binary semaphore.

- static void chBSemResetI (binary_semaphore_t *bsp, bool taken)
  
  Reset operation on the binary semaphore.

- static void chBSemReset (binary_semaphore_t *bsp, bool taken)
  
  Reset operation on the binary semaphore.

- static void chBSemSignalI (binary_semaphore_t *bsp)
  
  Performs a signal operation on a binary semaphore.

- static void chBSemSignal (binary_semaphore_t *bsp)
  
  Performs a signal operation on a binary semaphore.

- static bool chBSemGetStateI (const binary_semaphore_t *bsp)
  
  Returns the binary semaphore current state.

9.3.1 Detailed Description

Binary semaphores structures and macros.
Binary semaphores related APIs and services.
Operation mode

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

- **Taken**, when its counter has a value of zero or lower than zero. A negative number 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.

```c
#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 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)`

ChibiOS/RT
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) (_THREADS_QUEUE_DATA(name.queue))
  
  Data part of a static condition variable initializer.

- #define CONDVAR_DECL(name) condition_variable_t name = _CONDVAR_DATA(name)
  
  Static condition variable initializer.

Typedefs

- typedef struct condition_variable condition_variable_t
  
  condition_variable_t structure.

Functions

- void chCondObjectInit (condition_variable_t *cp)
  
  Initializes a condition_variable_t structure.

- void chCondSignal (condition_variable_t *cp)
  
  Signals one thread that is waiting on the condition variable.

- void chCondSignalI (condition_variable_t *cp)
  
  Signals one thread that is waiting on the condition variable.

- void chCondBroadcast (condition_variable_t *cp)
  
  Signals all threads that are waiting on the condition variable.

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

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

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

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

- msg_t chCondWaitTimeoutS (condition_variable_t *cp, sysinterval_t timeout)
  
  Waits on the condition variable releasing the mutex lock.
9.6.1 Detailed Description
Condition Variables macros and structures.

9.7 chconf.h File Reference
Configuration file template.

Macros

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

Kernel parameters and options
- \#define CH_CFG_TIME_QUANTUM 0
  Round robin interval.
- \#define CH_CFG_MEMCORE_SIZE 0
  Managed RAM size.
- \#define CH_CFG_NO_IDLE_THREAD FALSE
  Idle thread automatic spawn suppression.

Performance options
- \#define CH_CFG_OPTIMIZE_SPEED TRUE
  OS optimization.

Subsystem options
- \#define CH_CFG_USE_TM TRUE
  Time Measurement APIs.
- \#define CH_CFG_USE_REGISTRY TRUE
  Threads registry APIs.
- \#define CH_CFG_USE_WAITEXIT TRUE
  Threads synchronization APIs.
- \#define CH_CFG_USE_SEMAPHORES TRUE
  Semaphores APIs.
- \#define CH_CFG_USE_SEMAPHORES_PRIORITY FALSE
  Semaphores queuing mode.
- \#define CH_CFG_USE_MUTEXES TRUE
  Mutexes APIs.
- \#define CH_CFG_USE_MUTEXES_RECURSIVE FALSE
  Enables recursive behavior on mutexes.
- \#define CH_CFG_USE_CONDVARS TRUE
  Conditional Variables APIs.
- \#define CH_CFG_USE_CONDVARS_TIMEOUT TRUE
Conditional Variables APIs with timeout.
• #define CH_CFG_USE_EVENTS TRUE
  Events Flags APIs.
• #define CH_CFG_USE_EVENTS_TIMEOUT TRUE
  Events Flags APIs with timeout.
• #define CH_CFG_USE_MESSAGES TRUE
  Synchronous Messages APIs.
• #define CH_CFG_USE_MESSAGES_PRIORITY FALSE
  Synchronous Messages queuing mode.
• #define CH_CFG_USE_MAILBOXES TRUE
  Mailboxes APIs.
• #define CH_CFG_USE_MEMCORE TRUE
  Core Memory Manager APIs.
• #define CH_CFG_USE_HEAP TRUE
  Heap Allocator APIs.
• #define CH_CFG_USE_MEMPOOLS TRUE
  Memory Pools Allocator APIs.
• #define CH_CFG_USE_OBJ_FIFOS TRUE
  Objects FIFOs APIs.
• #define CH_CFG_USE_PIPES TRUE
  Pipes APIs.
• #define CH_CFG_USE_DYNAMIC TRUE
  Dynamic Threads APIs.

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

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

### Kernel hooks

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

### 9.7.1 Detailed Description

Configuration file template.
A copy of this file must be placed in each project directory, it contains the application specific kernel settings.

### 9.8 chcustomer.h File Reference

Customer-related info.

### Macros

• #define CH_CUSTOMER_ID_STRING "Santa, North Pole"
  Customer readable identifier.
• #define CH_CUSTOMER_ID_CODE "xxxx-yyyy"
  Customer code.
• #define CH_LICENSE CH_LICENSE_GPL
  Current license.
9.9 chdebug.c File Reference

Debug support code.

#include "ch.h"

Functions

- void _dbg_check_disable (void)
  Guard code for chSysDisable().
- void _dbg_check_suspend (void)
  Guard code for chSysSuspend().
- void _dbg_check_enable (void)
  Guard code for chSysEnable().
- void _dbg_check_lock (void)
  Guard code for chSysLock().
- void _dbg_check_unlock (void)
  Guard code for chSysUnlock().
- void _dbg_check_lock_from_isr (void)
  Guard code for chSysLockFromIsr().
- void _dbg_check_unlock_from_isr (void)
  Guard code for chSysUnlockFromIsr().
- void _dbg_check_enter_isr (void)
  Guard code for CH_IRQ_PROLOGUE().
- void _dbg_check_leave_isr (void)
  Guard code for CH_IRQ_EPILOGUE().
- void chDbgCheckClassI (void)
  I-class functions context check.
- void chDbgCheckClassS (void)
  S-class functions context check.

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_PORT_CM0 TRUE
- #define CH_CUSTOMER_LIC_PORT_CM3 TRUE
- #define CH_CUSTOMER_LIC_PORT_CM4 TRUE
- #define CH_CUSTOMER_LIC_PORT_CM7 TRUE
- #define CH_CUSTOMER_LIC_PORT_ARM79 TRUE
- #define CH_CUSTOMER_LIC_PORT_E200Z0 TRUE
- #define CH_CUSTOMER_LIC_PORT_E200Z2 TRUE
- #define CH_CUSTOMER_LIC_PORT_E200Z3 TRUE
- #define CH_CUSTOMER_LIC_PORT_E200Z4 TRUE
9.1 Detailed Description

Debug support code.

9.10 chdebug.h File Reference

Debug support macros and structures.

Macros

**Debug related settings**

- `#define CH_DBG_STACK_FILL_VALUE 0x55`
  
  Fill value for thread stack area in debug mode.

**Macro Functions**

- `#define chDbgCheck(c)`
  
  Function parameters check.
- `#define chDbgAssert(c, r)`
  
  Condition assertion.

9.10.1 Detailed Description

Debug support macros and structures.

9.11 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.

9.11.1 Detailed Description

Dynamic threads code.

9.12 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.

9.12.1 Detailed Description

Dynamic threads macros and structures.

9.13 chevents.c File Reference

Events code.

#include "ch.h"

Functions

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

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

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

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

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

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

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

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

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

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

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

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

- eventmask_t chEvtWaitOne (eventmask_t events)
  Waits for exactly one of the specified events.

- eventmask_t chEvtWaitAny (eventmask_t events)
Waits for any of the specified events.

• eventmask_t chEvtWaitAll (eventmask_t events)
  Waits for all the specified events.

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

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

9.13.1 Detailed Description

Events code.

9.14 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)((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(event_handler_t) (eventid_t id)
  Event Handler callback function.
Functions

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

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

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

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

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

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

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

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

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

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

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

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

- `eventmask_t chEvtWaitOne (eventmask_t events)`
  
  Waits for exactly one of the specified events.

- `eventmask_t chEvtWaitAny (eventmask_t events)`
  
  Waits for any of the specified events.

- `eventmask_t chEvtWaitAll (eventmask_t events)`
  
  Waits for all the specified events.

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

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

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

- `static void chEvtObjectInit (event_source_t *esp)`
  
  Initializes an Event Source.

- `static void chEvtRegisterMask (event_source_t *esp, event_listener_t *elp, eventmask_t events)`
  
  Registers an Event Listener on an Event Source.

- `static void chEvtRegister (event_source_t *esp, event_listener_t *elp, eventid_t event)`
  
  Registers an Event Listener on an Event Source.

- `static bool chEvtIsListeningI (event_source_t *esp)`
  
  Verifies if there is at least one event_listener_t registered.

- `static void chEvtBroadcast (event_source_t *esp)`
  
  Signals all the Event Listeners registered on the specified Event Source.

- `static void chEvtBroadcastI (event_source_t *esp)`
Signals all the Event Listeners registered on the specified Event Source.

- static eventmask_t chEvtAddEventsI (eventmask_t events)
  Adds (OR) a set of events to the current thread, this is much faster than using chEvtBroadcast() or chEvt←Signal().

- static eventmask_t chEvtGetEventsX (void)
  Returns the events mask.

9.14.1 Detailed Description

Events macros and structures.

9.15 chfactory.c File Reference

ChibiOS objects factory and registry code.

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

Functions

- void _factory_init (void)
  Initializes the objects factory.

- registered_object_t * chFactoryRegisterObject (const char *name, void *objp)
  Registers a generic object.

- registered_object_t * chFactoryFindObject (const char *name)
  Retrieves a registered object.

- registered_object_t * chFactoryFindObjectByPointer (void *objp)
  Retrieves a registered object by pointer.

- void chFactoryReleaseObject (registered_object_t *rop)
  Releases a registered object.

- dyn_buffer_t * chFactoryCreateBuffer (const char *name, size_t size)
  Creates a generic dynamic buffer object.

- dyn_buffer_t * chFactoryFindBuffer (const char *name)
  Retrieves a dynamic buffer object.

- void chFactoryReleaseBuffer (dyn_buffer_t *dbp)
  Releases a dynamic buffer object.

- dyn_semaphore_t * chFactoryCreateSemaphore (const char *name, cnt_t n)
  Creates a dynamic semaphore object.

- dyn_semaphore_t * chFactoryFindSemaphore (const char *name)
  Retrieves a dynamic semaphore object.

- void chFactoryReleaseSemaphore (dyn_semaphore_t *dsp)
  Releases a dynamic semaphore object.

- dyn_mailbox_t * chFactoryCreateMailbox (const char *name, size_t n)
  Creates a dynamic mailbox object.

- dyn_mailbox_t * chFactoryFindMailbox (const char *name)
  Retrieves a dynamic mailbox object.

- void chFactoryReleaseMailbox (dyn_mailbox_t *dmp)
  Releases a dynamic mailbox object.

- dyn_objects_fifo_t * chFactoryCreateObjectsFIFO (const char *name, size_t objsize, size_t objn, unsigned objalign)
  Creates a dynamic objects FIFO.
Creates a dynamic "objects FIFO" object.

- **dyn_objects_fifo_t ** chFactoryFindObjectFIFO (const char *name)
  Retrieves a dynamic "objects FIFO" object.

- void chFactoryReleaseObjectsFIFO (dyn_objects_fifo_t *dofp)
  Releases a dynamic "objects FIFO" object.

- **dyn_pipe_t ** chFactoryCreatePipe (const char *name, size_t size)
  Creates a dynamic pipe object.

- **dyn_pipe_t ** chFactoryFindPipe (const char *name)
  Retrieves a dynamic pipe object.

- void chFactoryReleasePipe (dyn_pipe_t *dpp)
  Releases a dynamic pipe object.

**Variables**

- **objects_factory_t ch_factory**
  Factory object static instance.

9.15.1 Detailed Description

ChibiOS objects factory and registry code.

9.16 chfactory.h File Reference

ChibiOS objects factory structures and macros.

**Data Structures**

- **struct ch_dyn_element**
  Type of a dynamic object list element.

- **struct ch_dyn_list**
  Type of a dynamic object list.

- **struct ch_registered_static_object**
  Type of a registered object.

- **struct ch_dyn_object**
  Type of a dynamic buffer object.

- **struct ch_dyn_semaphore**
  Type of a dynamic semaphore.

- **struct ch_dyn_mailbox**
  Type of a dynamic buffer object.

- **struct ch_dyn_objects_fifo**
  Type of a dynamic buffer object.

- **struct ch_dyn_pipe**
  Type of a dynamic pipe object.

- **struct ch_objects_factory**
  Type of the factory main object.
Macros

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

Typedefs

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

- void _factory_init (void)
  Initializes the objects factory.
- registered_object_t * chFactoryRegisterObject (const char *name, void *objp)
  Registers a generic object.
- registered_object_t * chFactoryFindObject (const char *name)
  Retrieves a registered object.
- registered_object_t * chFactoryFindObjectByPointer (void *objp)
  Retrieves a registered object by pointer.
- void chFactoryReleaseObject (registered_object_t *rop)
  Releases a registered object.
- dyn_buffer_t * chFactoryCreateBuffer (const char *name, size_t size)
  Creates a generic dynamic buffer object.
- dyn_buffer_t * chFactoryFindBuffer (const char *name)
  Retrieves a dynamic buffer object.
- void chFactoryReleaseBuffer (dyn_buffer_t *dbp)
  Releases a dynamic buffer object.
- dyn_semaphore_t * chFactoryCreateSemaphore (const char *name, cnt_t n)
  Creates a dynamic semaphore object.
- dyn_semaphore_t * chFactoryFindSemaphore (const char *name)
  Retrieves a dynamic semaphore object.
- void chFactoryReleaseSemaphore (dyn_semaphore_t *dsp)
  Releases a dynamic semaphore object.
- dyn_mailbox_t * chFactoryCreateMailbox (const char *name, size_t n)
  Creates a dynamic mailbox object.
- dyn_mailbox_t * chFactoryFindMailbox (const char *name)
  Retrieves a dynamic mailbox object.
- void chFactoryReleaseMailbox (dyn_mailbox_t *dmp)
  Releases a dynamic mailbox object.
- dyn_objects_fifo_t * chFactoryCreateObjectsFIFO (const char *name, size_t objsize, size_t objn, unsigned objalign)
  Creates a dynamic "objects FIFO" object.
- dyn_objects_fifo_t * chFactoryFindObjectByPointer (const char *name, size_t objsize, size_t objn, unsigned objalign)
  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.

9.16.1 Detailed Description

ChibiOS objects factory structures and macros.

9.17 chlib.h File Reference

ChibiOS/LIB main include file.
#include "chbsem.h"
#include "chmboxes.h"
#include "chmemcore.h"
#include "chmemheaps.h"
#include "chmempools.h"
#include "chobjfifos.h"
#include "chpipes.h"
#include "chfactory.h"

Macros

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

ChibiOS/LIB version identification

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

9.17.1 Detailed Description

ChibiOS/LIB main include file.
This header includes all the required library headers. This file is meant to be included by ch.h not directly by user.
License Module macros and structures.

```c
#include "chcustomer.h"
```

**Macros**

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

**Allowed Features Levels**

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

**Deployment Options**

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

**Licensing Options**

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

9.18.1 Detailed Description

License Module macros and structures.

9.19 chmboxes.c File Reference

Mailboxes code.

```c
#include "ch.h"
```
Functions

- void chMBObjectInit (mailbox_t *mbp, msg_t *buf, size_t n)
  
  Initializes a mailbox_t object.

- void chMBReset (mailbox_t *mbp)
  
  Resets a mailbox_t object.

- void chMBResetI (mailbox_t *mbp)
  
  Resets a mailbox_t object.

- msg_t chMBPostTimeout (mailbox_t *mbp, msg_t msg, sysinterval_t timeout)
  
  Posts a message into a mailbox.

- msg_t chMBPostTimeoutS (mailbox_t *mbp, msg_t msg, sysinterval_t timeout)
  
  Posts a message into a mailbox.

- msg_t chMBPostI (mailbox_t *mbp, msg_t msg)
  
  Posts a message into a mailbox.

- msg_t chMBPostAheadTimeout (mailbox_t *mbp, msg_t msg, sysinterval_t timeout)
  
  Posts an high priority message into a mailbox.

- msg_t chMBPostAheadTimeoutS (mailbox_t *mbp, msg_t msg, sysinterval_t timeout)
  
  Posts an high priority message into a mailbox.

- msg_t chMBPostAheadI (mailbox_t *mbp, msg_t msg)
  
  Posts an high priority message into a mailbox.

- msg_t chMBFetchTimeout (mailbox_t *mbp, msg_t *msgp, sysinterval_t timeout)
  
  Retrieves a message from a mailbox.

- msg_t chMBFetchTimeoutS (mailbox_t *mbp, msg_t *msgp, sysinterval_t timeout)
  
  Retrieves a message from a mailbox.

- msg_t chMBFetchI (mailbox_t *mbp, msg_t *msgp)
  
  Retrieves a message from a mailbox.

9.19.1 Detailed Description

Mailboxes code.

9.20 chmboxes.h File Reference

Mailboxes macros and structures.

Data Structures

- struct mailbox_t
  
  Structure representing a mailbox object.

Macros

- #define MAILBOX_DATA(name, buffer, size)
  
  Data part of a static mailbox initializer.

- #define MAILBOX_DECL(name, buffer, size) mailbox_t name = MAILBOX_DATA(name, buffer, size)
  
  Static mailbox initializer.
Functions

- void chMObjectInit (mailbox_t *mbp, msg_t *buf, size_t n)
  
  Initializes a mailbox_t object.

- void chMBReset (mailbox_t *mbp)
  
  Resets a mailbox_t object.

- void chMBResetI (mailbox_t *mbp)
  
  Resets a mailbox_t object.

- msg_t chMBPostTimeout (mailbox_t *mbp, msg_t msg, sysinterval_t timeout)
  
  Posts a message into a mailbox.

- msg_t chMBPostTimeoutS (mailbox_t *mbp, msg_t msg, sysinterval_t timeout)
  
  Posts a message into a mailbox.

- msg_t chMBPostI (mailbox_t *mbp, msg_t msg)
  
  Posts a message into a mailbox.

- msg_t chMBPostAheadTimeout (mailbox_t *mbp, msg_t msg, sysinterval_t timeout)
  
  Posts a high priority message into a mailbox.

- msg_t chMBPostAheadTimeoutS (mailbox_t *mbp, msg_t msg, sysinterval_t timeout)
  
  Posts a high priority message into a mailbox.

- msg_t chMBPostAheadI (mailbox_t *mbp, msg_t msg)
  
  Posts a high priority message into a mailbox.

- msg_t chMBFetchTimeout (mailbox_t *mbp, msg_t *msgp, sysinterval_t timeout)
  
  Retrieves a message from a mailbox.

- msg_t chMBFetchTimeoutS (mailbox_t *mbp, msg_t *msgp, sysinterval_t timeout)
  
  Retrieves a message from a mailbox.

- msg_t chMBFetchI (mailbox_t *mbp, msg_t *msgp)
  
  Retrieves a message from a mailbox.

- static size_t chMBGetSizeI (const mailbox_t *mbp)
  
  Returns the mailbox buffer size as number of messages.

- static size_t chMBGetUsedCountI (const mailbox_t *mbp)
  
  Returns the number of used message slots into a mailbox.

- static size_t chMBGetFreeCountI (const mailbox_t *mbp)
  
  Returns the number of free message slots into a mailbox.

- static msg_t chMBPeekI (const mailbox_t *mbp)
  
  Returns the next message in the queue without removing it.

- static void chMBResumeX (mailbox_t *mbp)
  
  Terminates the reset state.

9.20.1 Detailed Description

Mailboxes macros and structures.

9.21 chmemcore.c File Reference

Core memory manager code.

#include "ch.h"
Functions

- void _core_init (void)
  Low level memory manager initialization.
- void * chCoreAllocAlignedWithOffset (size_t size, unsigned align, size_t offset)
  Allocates a memory block.
- void * chCoreAllocAlignedWithOffset (size_t size, unsigned align, size_t offset)
  Allocates a memory block.
- size_t chCoreGetStatusX (void)
  Core memory status.

Variables

- memcore_t ch_memcore
  Memory core descriptor.

9.21.1 Detailed Description
Core memory manager code.

9.22 chmemcore.h File Reference
Core memory manager macros and structures.

Data Structures

- struct memcore_t
  Type of memory core object.

Macros

- #define CH_CFG_MEMCORE_SIZE 0
  Managed RAM size.

Typedefs

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

Functions

- void _core_init (void)
  Low level memory manager initialization.
- void * chCoreAllocAlignedWithOffset (size_t size, unsigned align, size_t offset)
  Allocates a memory block.
- void * chCoreAllocAlignedWithOffset (size_t size, unsigned align, size_t offset)
Allocates a memory block.

- `size_t chCoreGetStatusX (void)`
  Core memory status.

- `static void * chCoreAllocAlignedI (size_t size, unsigned align)`
  Allocates a memory block.

- `static void * chCoreAllocAligned (size_t size, unsigned align)`
  Allocates a memory block.

- `static void * chCoreAllocI (size_t size)`
  Allocates a memory block.

- `static void * chCoreAlloc (size_t size)`
  Allocates a memory block.

### 9.22.1 Detailed Description

Core memory manager macros and structures.

### 9.23 chmemheaps.c File Reference

Memory heaps code.

```c
#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.

### 9.23.1 Detailed Description

Memory heaps code.

### 9.24 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.

9.24.1 Detailed Description

Memory heaps macros and structures.

9.25 chmempools.c File Reference

Memory Pools code.

#include "ch.h"
Functions

- void chPoolObjectInitAligned (memory_pool_t *mp, size_t size, unsigned align, memgetfunc_t provider)
  
  Initializes an empty memory pool.
- void chPoolLoadArray (memory_pool_t *mp, void *p, size_t n)
  
  Loads a memory pool with an array of static objects.
- void * chPoolAlloc (memory_pool_t *mp)
  
  Allocates an object from a memory pool.
- void * chPoolAllocTimeout (guarded_memory_pool_t *gmp, sysinterval_t timeout)
  
  Allocates an object from a guarded memory pool.
- void chPoolFree (memory_pool_t *mp, void *objp)
  
  Releases an object into a memory pool.
- void chPoolFree (guarded_memory_pool_t *gmp, void *objp)
  
  Releases an object into a guarded memory pool.

9.25.1 Detailed Description

Memory Pools code.

9.26 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_De-
  
  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 = _GUARD←EDMEMORYPOOL_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 * chPoolAllocI (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 chPoolFreeI (memory_pool_t *mp, void *objp)
  
  Releases an object into a memory pool.

• void chGuardedPoolObjectInitAligned (guarded_memory_pool_t *gmp, size_t size, unsigned align)
  
  Initializes an empty guarded memory pool.

• void chGuardedPoolLoadArray (guarded_memory_pool_t *gmp, void *p, size_t n)
  
  Loads a guarded memory pool with an array of static objects.

• void * chGuardedPoolAllocTimeoutS (guarded_memory_pool_t *gmp, sysinterval_t timeout)
  
  Allocates an object from a guarded memory pool.

• void * chGuardedPoolAllocTimeout (guarded_memory_pool_t *gmp, sysinterval_t timeout)
  
  Allocates an object from a guarded memory pool.

• void chGuardedPoolFree (guarded_memory_pool_t *gmp, void *objp)
  
  Releases an object from a guarded memory pool.

• static void chPoolObjectInit (memory_pool_t *mp, size_t size, memgetfunc_t provider)

  Initializes an empty memory pool.

• static void chPoolAdd (memory_pool_t *mp, void *objp)

  Adds an object to a memory pool.

• static void chPoolAddI (memory_pool_t *mp, void *objp)

  Adds an object to a memory pool.

• static void chGuardedPoolObjectInit (guarded_memory_pool_t *gmp, size_t size)

  Initializes an empty guarded memory pool.

• static cnt_t 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 from a guarded memory pool.

• static void chGuardedPoolFreeS (guarded_memory_pool_t *gmp, void *objp)

  Releases an object from a guarded memory pool.

• static void chGuardedPoolAdd (guarded_memory_pool_t *gmp, void *objp)

  Adds an object to a guarded memory pool.

• static void chGuardedPoolAddI (guarded_memory_pool_t *gmp, void *objp)

  Adds an object to a guarded memory pool.

• static void chGuardedPoolAddS (guarded_memory_pool_t *gmp, void *objp)

  Adds an object to a guarded memory pool.
9.26.1 Detailed Description

Memory Pools macros and structures.

9.27 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 * chMsgWait (void)
  Suspends the thread and waits for an incoming message.
- void chMsgRelease (thread_t *tp, msg_t msg)
  Releases a sender thread specifying a response message.

9.27.1 Detailed Description

Messages code.

9.28 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 * chMsgWait (void)
  Suspends the thread and waits for an incoming message.
- void chMsgRelease (thread_t *tp, msg_t msg)
  Releases a sender thread specifying a response 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.

9.28.1 Detailed Description

Messages macros and structures.
9.29  chmtx.c File Reference

Mutexes code.
#include "ch.h"

Functions

• void chMtxObjectInit (mutex_t *mp)
  Initializes s mutex_t structure.
• void chMtxLock (mutex_t *mp)
  Locks the specified mutex.
• void chMtxLockS (mutex_t *mp)
  Locks the specified mutex.
• bool chMtxTryLock (mutex_t *mp)
  Tries to lock a mutex.
• bool chMtxTryLockS (mutex_t *mp)
  Tries to lock a mutex.
• void chMtxUnlock (mutex_t *mp)
  Unlocks the specified mutex.
• void chMtxUnlockS (mutex_t *mp)
  Unlocks the specified mutex.
• void chMtxUnlockAllS (void)
  Unlocks all mutexes owned by the invoking thread.
• void chMtxUnlockAll (void)
  Unlocks all mutexes owned by the invoking thread.

9.29.1 Detailed Description

Mutexes code.

9.30  chmtx.h File Reference

Mutexes macros and structures.

Data Structures

• struct ch_mutex
  Mutex structure.

Macros

• #define _MUTEX_DATA(name) { _THREADS_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.
9.31 chobjfifos.h File Reference

Objects FIFO structures and macros.

Data Structures

- struct ch_objects_fifo
  Type of an objects FIFO.

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.

9.30.1 Detailed Description

Mutexes macros and structures.

9.31 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.

9.31.1 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.
9.32 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.

9.32.1 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.

9.33 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.

9.33.1 Detailed Description

Pipes macros and structures.

9.34 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)`
  
  Returns the thread with the specified name.
```
Retrieves a thread pointer by name.

- \texttt{thread\_t \* chRegFindThreadByPointer (thread\_t \*tp)}
  
  Confirms that a pointer is a valid thread pointer.

- \texttt{thread\_t \* chRegFindThreadByWorkingArea (stkalign\_t \*wa)}
  
  Confirms that a working area is being used by some active thread.

9.34.1 Detailed Description

Threads registry code.

9.35 \texttt{chregistry.h} File Reference

Threads registry macros and structures.

Data Structures

- \texttt{struct chdebug\_t}
  
  ChibiOS/RT memory signature record.

Macros

- \texttt{#define REG\_REMOVE(tp)}
  
  Removes a thread from the registry list.

- \texttt{#define REG\_INSERT(tp)}
  
  Adds a thread to the registry list.

Functions

- \texttt{thread\_t \* chRegFirstThread (void)}
  
  Returns the first thread in the system.

- \texttt{thread\_t \* chRegNextThread (thread\_t \*tp)}
  
  Returns the thread next to the specified one.

- \texttt{thread\_t \* chRegFindThreadByName (const char \*name)}
  
  Retrieves a thread pointer by name.

- \texttt{thread\_t \* chRegFindThreadByPointer (thread\_t \*tp)}
  
  Confirms that a pointer is a valid thread pointer.

- \texttt{thread\_t \* chRegFindThreadByWorkingArea (stkalign\_t \*wa)}
  
  Confirms that a working area is being used by some active thread.

- \texttt{static void chRegSetThreadName (const char \*name)}
  
  Sets the current thread name.

- \texttt{static const char \* chRegGetThreadNameX (thread\_t \*tp)}
  
  Returns the name of the specified thread.

- \texttt{static void chRegSetThreadNameX (thread\_t \*tp, const char \*name)}
  
  Changes the name of the specified thread.

9.35.1 Detailed Description

Threads registry macros and structures.
9.36  chrestrictions.h File Reference

Licensing restrictions header.

9.36.1  Detailed Description

Licensing restrictions header.

9.37  chschd.c File Reference

Scheduler code.
#include "ch.h"

Functions

- void _scheduler_init (void)
  Scheduler initialization.
- void queue_prio_insert (thread_t *tp, threads_queue_t *tqp)
  Inserts a thread into a priority ordered queue.
- void queue_insert (thread_t *tp, threads_queue_t *tqp)
  Inserts a thread into a queue.
- thread_t * queue_fifo_remove (threads_queue_t *tqp)
  Removes the first-out thread from a queue and returns it.
- thread_t * queue_lifo_remove (threads_queue_t *tqp)
  Removes the last-out thread from a queue and returns it.
- thread_t * queue_dequeue (thread_t *tp)
  Removes a thread from a queue and returns it.
- void list_insert (thread_t *tp, threads_list_t *tlp)
  Pushes a thread_t on top of a stack list.
- thread_t * list_remove (threads_list_t *tlp)
  Pops a thread from the top of a stack list and returns it.
- thread_t * chSchReadyI (thread_t *tp)
  Inserts a thread in the Ready List placing it behind its peers.
- thread_t * chSchReadyAheadI (thread_t *tp)
  Inserts a thread in the Ready List placing it ahead its peers.
- void chSchGoSleepS (tstate_t newstate)
  Puts the current thread to sleep into the specified state.
- msg_t chSchGoSleepTimeoutS (tstate_t newstate, sysinterval_t timeout)
  Puts the current thread to sleep into the specified state with timeout specification.
- void chSchWakeupS (thread_t *ntp, msg_t msg)
  Wakes up a thread.
- void chSchRescheduleS (void)
  Performs a reschedule if a higher priority thread is runnable.
- bool chSchIsPreemptionRequired (void)
  Evaluates if preemption is required.
- void chSchDoRescheduleBehind (void)
  Switches to the first thread on the runnable queue.
- void chSchDoRescheduleAhead (void)
Switches to the first thread on the runnable queue.

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

Variables

- ch_system_t ch
  System data structures.

9.37.1 Detailed Description

Scheduler code.

9.38 chschd.h File Reference

Scheduler macros and structures.

Data Structures

- struct ch_threads_list
  Generic threads single link list, it works like a stack.
- struct ch_threads_queue
  Generic threads bidirectional linked list header and element.
- struct ch_thread
  Structure representing a thread.
- struct ch_virtual_timer
  Virtual Timer descriptor structure.
- struct ch_virtual_timers_list
  Virtual timers list header.
- struct ch_system_debug
  System debug data structure.
- struct ch_system
  System data structure.

Macros

- #define firstprio(rlp) ((rlp)->next->prio)
  Returns the priority of the first thread on the given ready list.
- #define currp ch.rlist.current
  Current thread pointer access macro.

Wakeup status codes

- #define MSG_OK (msg_t)0
  Normal wake up 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 NOPRI (tprio_t)0
  Ready list header priority.
- #define IDLEPRI (tprio_t)1
  Idle priority.
- #define LOWPRI (tprio_t)2
  Lowest priority.
- #define NORMALPRI (tprio_t)128
  Normal priority.
- #define HIGHPRI (tprio_t)255
  Highest priority.

Thread states

- #define CH_STATE_READY (tstate_t)0
  Waiting on the ready list.
- #define CH_STATE_CURRENT (tstate_t)1
  Currently running.
- #define CH_STATE_WTSTART (tstate_t)2
  Just created.
- #define CH_STATE_SUSPENDED (tstate_t)3
  Suspended state.
- #define CH_STATE_QUEUE (tstate_t)4
  On a queue.
- #define CH_STATE_WTSEM (tstate_t)5
  On a semaphore.
- #define CH_STATE_WTMTX (tstate_t)6
  On a mutex.
- #define CH_STATE_WTCOND (tstate_t)7
  On a cond variable.
- #define CH_STATE_SLEEPING (tstate_t)8
  Sleeping.
- #define CH_STATE_WTEXIT (tstate_t)9
  Waiting a thread.
- #define CH_STATE_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

- void _scheduler_init (void)
  
  Scheduler initialization.

- thread_t * chSchReadyI (thread_t *tp)
  
  Inserts a thread in the Ready List placing it behind its peers.

- thread_t * chSchReadyAheadI (thread_t *tp)
  
  Inserts a thread in the Ready List placing it ahead its peers.

- void chSchGoSleepS (tstate_t newstate)
  
  Puts the current thread to sleep into the specified state.

- msg_t chSchGoSleepTimeoutS (tstate_t newstate, sysinterval_t timeout)
  
  Puts the current thread to sleep into the specified state with timeout specification.

- void chSchWakeupS (thread_t *ntp, msg_t msg)
  
  Wakes up a thread.

- void chSchRescheduleS (void)
  
  Performs a reschedule if a higher priority thread is runnable.

- bool chSchIsPreemptionRequired (void)
  
  Evaluates if preemption is required.

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

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

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

- void queue_prio_insert (thread_t *tp, threads_queue_t *tqp)
  
  Inserts a thread into a priority ordered queue.

- void queue_insert (thread_t *tp, threads_queue_t *tqp)
  
  Inserts a thread into a queue.

- thread_t * queue_fifo_remove (threads_queue_t *tqp)
  
  Removes the first-out thread from a queue and returns it.

- thread_t * queue_lifo_remove (threads_queue_t *tqp)
  
  Removes the last-out thread from a queue and returns it.

- thread_t * queue_dequeue (thread_t *tp)
  
  Removes a thread from a queue and returns it.

- void list_insert (thread_t *tp, threads_list_t *tlp)
  
  Pushes a thread_t on top of a stack list.

- thread_t * list_remove (threads_list_t *tlp)
  
  Pops a thread from the top of a stack list and returns it.

- static void list_init (threads_list_t *tlp)
  
  Threads list initialization.

- static bool list_isempty (threads_list_t *tlp)
  
  Evaluates to true if the specified threads list is empty.

- static bool list_notempty (threads_list_t *tlp)
  
  Evaluates to true if the specified threads list is not empty.

- static void queue_init (threads_queue_t *tqp)
  
  Threads queue initialization.

- static bool queue_isempty (const threads_queue_t *tqp)
  
  Evaluates to true if the specified threads queue is empty.

- static bool queue_notempty (const threads_queue_t *tqp)
  
  Evaluates to true if the specified threads queue is not empty.
Determine if the current thread must reschedule.

- static bool chSchCanYieldS (void)
  Determines if yielding is possible.

- static void chSchDoYieldS (void)
  Yields the time slot.

- static void chSchPreemption (void)
  Inline-able preemption code.

9.38.1 Detailed Description

Scheduler macros and structures.

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

- void chSemResetI (semaphore_t *sp, cnt_t n)
  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.

9.39.1 Detailed Description

Semaphores code.
Semaphores macros and structures.

Data Structures

- `struct ch_semaphore`
  Semaphore structure.

Macros

- `#define _SEMAPHORE_DATA(name, n) (_THREADS_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 chSemReset (semaphore_t *sp, cnt_t n)`
  Performs a reset operation on the semaphore.
- `void chSemResetI (semaphore_t *sp, cnt_t n)`
  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 chSemFastWaitI (semaphore_t *sp)`
  Decreases the semaphore counter.
- `static void chSemFastSignall (semaphore_t *sp)`
  Increases the semaphore counter.
- `static cnt_t chSemGetCounterI (const semaphore_t *sp)`
  Returns the semaphore counter current value.
9.40.1 Detailed Description

Semaphores macros and structures.

9.41 chstats.c File Reference

Statistics module code.
#include "ch.h"

Functions

- void _stats_init (void)
  Initializes the statistics module.
- void _stats_increase_irq (void)
  Increases the IRQ counter.
- void _stats_ctxswc (thread_t *ntp, thread_t *otp)
  Updates context switch related statistics.
- void _stats_start_measure_crit_thd (void)
  Starts the measurement of a thread critical zone.
- void _stats_stop_measure_crit_thd (void)
  Stops the measurement of a thread critical zone.
- void _stats_start_measure_crit_isr (void)
  Starts the measurement of an ISR critical zone.
- void _stats_stop_measure_crit_isr (void)
  Stops the measurement of an ISR critical zone.

9.41.1 Detailed Description

Statistics module code.

9.42 chstats.h File Reference

Statistics module macros and structures.

Data Structures

- struct kernel_stats_t
  Type of a kernel statistics structure.

Functions

- void _stats_init (void)
  Initializes the statistics module.
- void _stats_increase_irq (void)
  Increases the IRQ counter.
- void _stats_ctxswc (thread_t *ntp, thread_t *otp)
  Updates context switch related statistics.
9.43 chsys.c File Reference

System related code.

#include "ch.h"

Functions

• THD_WORKING_AREA (ch_idle_thread_wa, PORT_IDLE_THREAD_STACK_SIZE)
  Idle thread working area.
• static void _idle_thread (void *) p)
  This function implements the idle thread infinite loop.
• void chSysInit (void)
  ChibiOS/RT initialization.
• void chSysHalt (const char * reason)
  Halts the system.
• bool chSysIntegrityCheckI (unsigned testmask)
  System integrity check.
• void chSysTimerHandlerI (void)
  Handles time ticks for round robin preemption and timer increments.
• syssts_t chSysGetStatusAndLockX (void)
  Returns the execution status and enters a critical zone.
• void chSysRestoreStatusX (syssts_t sts)
  Restores the specified execution status and leaves a critical zone.
• bool chSysIsCounterWithinX (rtcnt_t cnt, rtcnt_t start, rtcnt_t end)
  Realtime window test.
• void chSysPolledDelayX (rtcnt_t cycles)
  Polled delay.

9.43.1 Detailed Description

System related code.

9.44 chsys.h File Reference

System related macros and structures.
Macros

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

Masks of executable integrity checks.

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

ISRs abstraction macros

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

Fast ISRs abstraction macros

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

Time conversion utilities for the realtime counter

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

Functions

- `void chSysInit (void)`
  ChibiOS/RT initialization.
- `bool chSysIntegrityCheckI (unsigned testmask)`
  System integrity check.
- `void chSysTimerHandlerI (void)`
  Handles time ticks for round robin preemption and timer increments.
9.45 chsystypes.h File Reference

System types header.

Macros

- `#define __CH_STRINGIFY(a) #a`
  Utility to make the parameter a quoted string.

Typedefs

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

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

- `typedef struct ch_threads_list threads_list_t`
  ChibiOS/RT
Type of a generic threads single link list, it works like a stack.
• typedef struct ch_threads_queue threads_queue_t
  Type of a generic threads bidirectional linked list header and element.
• typedef struct ch_ready_list ready_list_t
  Type of a ready list header.
• typedef void(∗ vfunc_t) (void ∗p)
  Type of a Virtual Timer callback function.
• typedef struct ch_virtual_timer virtual_timer_t
  Type of a Virtual Timer structure.
• typedef struct ch_virtual_timers_list virtual_timers_list_t
  Type of virtual timers list header.
• typedef struct ch_system_debug system_debug_t
  Type of a system debug structure.
• typedef struct ch_system ch_system_t
  Type of system data structure.

9.45.1 Detailed Description

System types header.

9.46 chthreads.c File Reference

Threads code.

#include "ch.h"

Functions

• thread_t ∗_thread_init (thread_t ∗tp, const char ∗name, tprio_t prio)
  Initializes a thread structure.
• void _thread_memfill (uint8_t ∗startp, uint8_t ∗endp, uint8_t ∗v)
  Memory fill utility.
• thread_t ∗chThdCreateSuspendedI (const thread_descriptor_t ∗tdp)
  Creates a new thread into a static memory area.
• thread_t ∗chThdCreateSuspended (const thread_descriptor_t ∗tdp)
  Creates a new thread into a static memory area.
• thread_t ∗chThdCreateI (const thread_descriptor_t ∗tdp)
  Creates a new thread into a static memory area.
• thread_t ∗chThdCreate (const thread_descriptor_t ∗tdp)
  Creates a new thread into a static memory area.
• 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)
9.47 chthreads.h File Reference 461

9.47.1 Detailed Description

Threads code.

9.47 chthreads.h File Reference

Threads module macros and structures.

Data Structures

• struct thread_descriptor_t
  Type of a thread descriptor.
Macros

Threads queues

• #define _THREADS_QUEUE_DATA(name) {{thread_t *} &name, {thread_t *} &name}
  Data part of a static threads queue object initializer.
• #define _THREADS_QUEUEDECL(name) threads_queue_t name = _THREADS_QUEUE_DATA(name)
  Static threads queue object initializer.

Working Areas

• #define THD_WORKING_AREA_SIZE(n) MEM_ALIGN_NEXT(sizeof(thread_t) + PORT_WA_SIZE(n), PORT_STACK_ALIGN)
  Calculates the total Working Area size.
• #define THD_WORKING_AREA(s, n) PORT_WORKING_AREA(s, n)
  Static working area allocation.
• #define THD_WORKING_AREA_BASE(s) ((stkalign_t *) (s))
  Base of a working area casted to the correct type.
• #define THD_WORKING_AREA_END(s)
  End of a working area casted to the correct type.

Threads abstraction macros

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

Macro Functions

• #define chThdSleepSeconds(sec) chThdSleep(TIME_S2I(sec))
  Delays the invoking thread for the specified number of seconds.
• #define chThdSleepMilliseconds(msec) chThdSleep(TIME_MS2I(msec))
  Delays the invoking thread for the specified number of milliseconds.
• #define chThdSleepMicroseconds(usec) chThdSleep(TIME_US2I(usec))
  Delays the invoking thread for the specified number of microseconds.

Typedefs

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

Functions

• thread_t * _thread_init (thread_t *tp, const char *name, tprio_t prio)
  Initializes a thread structure.
• void _thread_memfill (uint8_t * startp, uint8_t * endp, uint8_t * v)
  Memory fill utility.
• thread_t * 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)
9.47 chthreads.h File Reference

Creates a new thread into a static memory area.

- `thread_t * chThdStart (thread_t *tp)`
  - Resumes a thread created with `chThdCreateI()`.
- `thread_t * chThdAddRef (thread_t *tp)`
  - Adds a reference to a thread object.
- `void chThdRelease (thread_t *tp)`
  - Releases a reference to a thread object.
- `void chThdExit (msg_t msg)`
  - Terminates the current thread.
- `void chThdExitS (msg_t msg)`
  - Terminates the current thread.
- `msg_t chThdWait (thread_t *tp)`
  - Blocks the execution of the invoking thread until the specified thread terminates then the exit code is returned.
- `tprio_t chThdSetPriority (tprio_t newprio)`
  - Changes the running thread priority level then reschedules if necessary.
- `void chThdTerminate (thread_t *tp)`
  - Requests a thread termination.
- `msg_t chThdSuspendS (thread_reference_t *trp)`
  - Sends the current thread sleeping and sets a reference variable.
- `msg_t chThdSuspendTimeoutS (thread_reference_t *trp, sysinterval_t timeout)`
  - Sends the current thread sleeping and sets a reference variable.
- `void chThdResumeI (thread_reference_t *trp, msg_t msg)`
  - Wakes up a thread waiting on a thread reference object.
- `void chThdResumeS (thread_reference_t *trp, msg_t msg)`
  - Wakes up a thread waiting on a thread reference object.
- `void chThdResume (thread_reference_t *trp, msg_t msg)`
  - Wakes up a thread waiting on a thread reference object.
- `msg_t chThdEnqueueTimeoutS (threads_queue_t *tqp, sysinterval_t timeout)`
  - Enqueues the caller thread on a threads queue object.
- `void 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.
- `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.

ChibiOS/RT
464 File Documentation

- static bool chThdShouldTerminateX (void)
  Verifies if the current thread has a termination request pending.
- static thread_t * chThdStartI (thread_t *tp)
  Resumes a thread created with chThdCreateI().
- static void chThdSleepS (sysinterval_t ticks)
  Suspends the invoking thread for the specified number of ticks.
- static void chThdQueueObjectInit (threads_queue_t *tqp)
  Initializes a threads queue object.
- static bool chThdQueueIsEmptyI (threads_queue_t *tqp)
  Evaluates to true if the specified queue is empty.
- static void chThdDoDequeueNextI (threads_queue_t *tqp, msg_t msg)
  Dequeues and wakes up one thread from the threads queue object.

9.47.1 Detailed Description

Threads module macros and structures.

9.48 chtime.h File Reference

Time and intervals macros and structures.

Macros

Special time constants

- #define TIME_IMMEDIATE ((sysinterval_t)0)
  Zero interval specification for some functions with a timeout specification.
- #define TIME_INFINITE ((sysinterval_t)-1)
  Infinite interval specification for all functions with a timeout specification.
- #define TIME_MAX_INTERVAL ((sysinterval_t)-2)
  Maximum interval constant usable as timeout.
- #define TIME_MAX_SYSTIME ((systime_t)-1)
  Maximum system of system time before it wraps.

Fast time conversion utilities

- #define TIME_S2I(secs) ((sysinterval_t)((time_conv_t)(secs) ∗ (time_conv_t)CH_CFG_ST_FREQUEN→CY))
  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

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

Functions

**Secure time conversion utilities**

- \texttt{static sysinterval_t chTimeS2I (time_secs_t secs)}
  Seconds to time interval.
- \texttt{static sysinterval_t chTimeMS2I (time_msecs_t msec)}
  Milliseconds to time interval.
- \texttt{static sysinterval_t chTimeUS2I (time_usecs_t usec)}
  Microseconds to time interval.
- \texttt{static time_secs_t chTimeI2S (sysinterval_t interval)}
  Time interval to seconds.
- \texttt{static time_msecs_t chTimeI2MS (sysinterval_t interval)}
  Time interval to milliseconds.
- \texttt{static time_usecs_t chTimeI2US (sysinterval_t interval)}
  Time interval to microseconds.
- \texttt{static systime_t chTimeAddX (systime_t systime, sysinterval_t interval)}
  Adds an interval to a system time returning a system time.
- \texttt{static sysinterval_t chTimeDiffX (systime_t start, systime_t end)}
  Subtracts two system times returning an interval.
- \texttt{static bool chTimeIsInRangeX (systime_t time, systime_t start, systime_t end)}
  Checks if the specified time is within the specified time range.

9.48.1 Detailed Description

Time and intervals macros and structures.

9.49 chtm.c File Reference

Time Measurement module code.

\#include "ch.h"
Functions

- void _tm_init (void)
  *Initializes the time measurement unit.*
- void chTMObjectInit (time_measurement_t *tmp)
  *Initializes a TimeMeasurement object.*
- NOINLINE void chTMStartMeasurementX (time_measurement_t *tmp)
  *Starts a measurement.*
- NOINLINE void chTMStopMeasurementX (time_measurement_t *tmp)
  *Stops a measurement.*
- NOINLINE void chTMChainMeasurementToX (time_measurement_t *tmp1, time_measurement_t *tmp2)
  *Stops a measurement and chains to the next one using the same time stamp.*

9.49.1 Detailed Description

Time Measurement module code.

9.50 chtm.h File Reference

Time Measurement module macros and structures.

Data Structures

- struct tm_calibration_t
  *Type of a time measurement calibration data.*
- struct time_measurement_t
  *Type of a Time Measurement object.*

Functions

- void _tm_init (void)
  *Initializes the time measurement unit.*
- void chTMObjectInit (time_measurement_t *tmp)
  *Initializes a TimeMeasurement object.*
- NOINLINE void chTMStartMeasurementX (time_measurement_t *tmp)
  *Starts a measurement.*
- NOINLINE void chTMStopMeasurementX (time_measurement_t *tmp)
  *Stops a measurement.*
- NOINLINE void chTMChainMeasurementToX (time_measurement_t *tmp1, time_measurement_t *tmp2)
  *Stops a measurement and chains to the next one using the same time stamp.*

9.50.1 Detailed Description

Time Measurement module macros and structures.
9.51 chtrace.c File Reference

Tracer code.

#include "ch.h"

Functions

- static NOINLINE void trace_next (void)
  
  Writes a time stamp and increases the trace buffer pointer.

- void _trace_init (void)
  
  Trace circular buffer subsystem initialization.

- void _trace_switch (thread_t *ntp, thread_t *otp)
  
  Inserts in the circular debug trace buffer a context switch record.

- void _trace_isr_enter (const char *isr)
  
  Inserts in the circular debug trace buffer an ISR-enter record.

- void _trace_isr_leave (const char *isr)
  
  Inserts in the circular debug trace buffer an ISR-leave record.

- void _trace_halt (const char *reason)
  
  Inserts in the circular debug trace buffer an halt record.

- void chDbgWriteTraceI (void *up1, void *up2)
  
  Adds an user trace record to the trace buffer.

- void chDbgWriteTrace (void *up1, void *up2)
  
  Adds an user trace record to the trace buffer.

- void chDbgSuspendTraceI (uint16_t mask)
  
  Suspends one or more trace events.

- void chDbgSuspendTrace (uint16_t mask)
  
  Suspends one or more trace events.

- void chDbgResumeTraceI (uint16_t mask)
  
  Resumes one or more trace events.

- void chDbgResumeTrace (uint16_t mask)
  
  Resumes one or more trace events.

9.51.1 Detailed Description

Tracer code.

9.52 chtrace.h File Reference

Tracer macros and structures.

Data Structures

- struct ch_trace_event_t
  
  Trace buffer record.

- struct ch_trace_buffer_t
  
  Trace buffer header.
Macros

Trace record types

- #define CH_TRACE_TYPE_UNUSED 0U
- #define CH_TRACE_TYPE_SWITCH 1U
- #define CH_TRACE_TYPE_ISR_ENTER 2U
- #define CH_TRACE_TYPE_ISR_LEAVE 3U
- #define CH_TRACE_TYPE_HALT 4U
- #define CH_TRACE_TYPE_USER 5U

Events to trace

- #define CH_DBG_TRACE_MASK_DISABLED 255U
- #define CH_DBG_TRACE_MASK_NONE 0U
- #define CH_DBG_TRACE_MASK_SWITCH 1U
- #define CH_DBG_TRACE_MASK_ISR 2U
- #define CH_DBG_TRACE_MASK_HALTS 4U
- #define CH_DBG_TRACE_MASK_USER 8U
- #define CH_DBG_TRACE_MASK_SLOW
- #define CH_DBG_TRACE_MASK_ALL

Debug related settings

- #define CH_DBG_TRACE_MASK CH_DBG_TRACE_MASK_DISABLED
- #define CH_DBG_TRACE_BUFFER_SIZE 128

Functions

- void _trace_switch (thread_t *ntp, thread_t *otp)
  Inserts in the circular debug trace buffer a context switch record.
- void _trace_isr_enter (const char *isr)
  Inserts in the circular debug trace buffer an ISR-enter record.
- void _trace_isr_leave (const char *isr)
  Inserts in the circular debug trace buffer an ISR-leave record.
- void _trace_halt (const char *reason)
  Inserts in the circular debug trace buffer an halt record.
- void chDbgWriteTraceI (void *up1, void *up2)
  Adds an user trace record to the trace buffer.
- void chDbgWriteTrace (void *up1, void *up2)
  Adds an user trace record to the trace buffer.
- void chDbgSuspendTraceI (uint16_t mask)
  Suspends one or more trace events.
- void chDbgSuspendTrace (uint16_t mask)
  Suspends one or more trace events.
- void chDbgResumeTraceI (uint16_t mask)
  Resumes one or more trace events.
- void chDbgResumeTrace (uint16_t mask)
  Resumes one or more trace events.

9.52.1 Detailed Description

Tracer macros and structures.
9.53 chversion.h File Reference

Version Module macros and structures.

Macros

- `#define _CHIBIOS_`  
  ChibiOS product identification macro.
- `#define CH_VERSION_STABLE 1`  
  Stable release flag.

ChibiOS version identification

- `#define CH_VERSION "19.1.3"`  
  ChibiOS version string.
- `#define CH_VERSION_YEAR 19`  
  ChibiOS version release year.
- `#define CH_VERSION_MONTH 1`  
  ChibiOS version release month.
- `#define CH_VERSION_PATCH 3`  
  ChibiOS version patch number.
- `#define CH_VERSION_NICKNAME "Maiori"`  
  ChibiOS version nickname.

9.53.1 Detailed Description

Version Module macros and structures.

9.54 chvt.c File Reference

Time and Virtual Timers module code.

`#include "ch.h"`

Functions

- `void _vt_init (void)`  
  Virtual Timers initialization.
- `void chVTDoSetI (virtual_timer_t *vtp, sysinterval_t delay, vtfunc_t vtfunc, void *par)`  
  Enables a virtual timer.
- `void chVTDoResetI (virtual_timer_t *vtp)`  
  Disables a Virtual Timer.

9.54.1 Detailed Description

Time and Virtual Timers module code.

9.55 chvt.h File Reference

Time and Virtual Timers module macros and structures.
Functions

- **void _vt_init (void)**
  
  *Virtual Timers initialization.*

- **void chVTDoset (virtual_timer_t *vtp, sysinterval_t delay, vfunc_t vtfunc, void *par)**
  
  *Enables a virtual timer.*

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

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

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

- **static void chVTDoTickI (void)**
  
  *Virtual timers ticker.*

9.55.1 Detailed Description

Time and Virtual Timers module macros and structures.
Index

_BSEMAPHORE_DATA
  Binary Semaphores, 253

_CHIBIOS_
  Release Information, 20

_CHIBIOS_OSLIB_
  Version Numbers and Identification, 251

_CHIBIOS_RT_
  Version Numbers and Identification, 26

_CONDVAR_DATA
  Condition Variables, 184

_EVENTSOURCE_DATA
  Event Flags, 196

_GUARDEDMEMORYPOOL_DATA
  Memory Pools, 302

_MAILBOX_DATA
  Mailboxes, 264

_MEMORYPOOL_DATA
  Memory Pools, 301

_MUTEK_DATA
  Mutexes, 172

PIPE_DATA
  Pipes, 278

_SEMAPHORE_DATA
  Counting Semaphores, 157

_THREADS_QUEUE_DATA
  Threads, 119

_THREADS_QUEUE_DECL
  Threads, 119

__CH_STRINGIFY
  Scheduler, 74

_core_init
  Core Memory Manager, 288

_dbg_check_disable
  Checks and Assertions, 234

_dbg_check_enable
  Checks and Assertions, 234

_dbg_check_enter_isr
  Checks and Assertions, 236

_dbg_check_leave_isr
  Checks and Assertions, 237

_dbg_check_lock
  Checks and Assertions, 235

_dbg_check_lock_from_isr
  Checks and Assertions, 235

_dbg_check_suspend
  Checks and Assertions, 234

_dbg_check_unlock
  Checks and Assertions, 235

_dbg_check_unlock_from_isr
  Checks and Assertions, 235

_checks_and Assertions, 236

_factory_init
  Dynamic Objects Factory, 338

_heap_init
  Memory Heaps, 295

_idle_thread
  System Management, 56

_scheduler_init
  Scheduler, 75

_stats_ctxswc
  Statistics, 248

_stats_increase_irq
  Statistics, 247

_stats_init
  Statistics, 247

_stats_start_measure_crit_isr
  Statistics, 249

_stats_start_measure_crit_thd
  Statistics, 248

_stats_stop_measure_crit_isr
  Statistics, 249

_stats_stop_measure_crit_thd
  Statistics, 248

_thread_init
  Threads, 121

_thread_memfill
  Threads, 122

_tm_init
  Time Measurement, 152

_trace_halt
  Tracing, 242

_trace_init
  Tracing, 240

_trace_isr_enter
  Tracing, 241

_trace_isr_leave
  Tracing, 242

_trace_switch
  Tracing, 241

_vt_init
  Virtual Timers, 101

ALL_EVENTS
  Event Flags, 196

align
  memory_pool_t, 408

arg
  thread_descriptor_t, 413

BSEMAPHORE_DECL
Binary Semaphores, 254
Base Kernel Services, 46
best
time_measurement_t, 414
Binary Semaphores, 253
_BSEMAPHORE_DATA, 253
BSEMAPHORE_DECL, 254
binary_semaphore_t, 254
chBSemGetStateI, 261
chBSemObjectInit, 254
chBSemReset, 258
chBSemResetI, 258
chBSemSignal, 260
chBSemSignalI, 259
chBSemWait, 255
chBSemWaitTimeout, 257
chBSemWaitTimeoutS, 256
chBSemWaitS, 255
binary_semaphore_t
Binary Semaphores, 254
buf_list
ch_objects_factory, 366
buffer
ch_dyn_object, 357
ch_dyn_pipe, 361
ch_trace_buffer_t, 385
mailbox_t, 404
pipe_t, 410
CH_CFG_CONTEXT_SWITCH_HOOK
Options, 41
CH_CFG_FACTORY_GENERIC_BUFFERS
Dynamic Objects Factory, 336
Options, 38
CH_CFG_FACTORY_MAILBOXES
Dynamic Objects Factory, 336
Options, 38
CH_CFG_FACTORY_MAX_NAMES_LENGTH
Dynamic Objects Factory, 336
Options, 38
CH_CFG_FACTORY_OBJ_FIFOS
Dynamic Objects Factory, 336
Options, 38
CH_CFG_FACTORY_OBJECTS_REGISTRY
Dynamic Objects Factory, 336
Options, 38
CH_CFG_FACTORY_PIPES
Dynamic Objects Factory, 336, 337
Options, 39
CH_CFG_FACTORY_SEMAPHORES
Dynamic Objects Factory, 336
Options, 38
CH_CFG_IDLE_ENTER_HOOK
Options, 42
CH_CFG_IDLE_LEAVE_HOOK
Options, 42
CH_CFG_IDLE_LOOP_HOOK
Options, 42
CH_CFG_INTERVALS_SIZE
Options, 33
CH_CFG_IRQ_PROLOGUE_HOOK
Options, 41
CH_CFG_IRQ_EPILOGUE_HOOK
Options, 41
CH_CFG_MEMCORE_SIZE
Core Memory Manager, 288
Options, 33
CH_CFG_NO_IDLE_THREAD
Options, 34
CH_CFG_OPTIMIZE_SPEED
Options, 34
CH_CFG_ST_FREQUENCY
Options, 33
CH_CFG_ST_RESOLUTION
Options, 33
CH_CFG_ST_TIMEDELTA
Options, 33
CH_CFG_SYSTEM_EXTRA_FIELDS
Options, 40
CH_CFG_SYSTEM_HALTED_HOOK
Options, 42
CH_CFG_SYSTEM_INIT_HOOK
Options, 40
CH_CFG_SYSTEM_TICK_HOOK
Options, 42
CH_CFG_THREAD_EXIT_HOOK
Options, 41
CH_CFG_THREAD_EXTRA_FIELDS
Options, 41
CH_CFG_THREAD_INIT_HOOK
Options, 41
CH_CFG_TIME_QUANTUM
Options, 33
CH_CFG_TIME_TYPES_SIZE
Options, 33
CH_CFG_TRACE_HOOK
Options, 33
CH_CFG_USE_CONDVARS_TIMEOUT
Options, 43
CH_CFG_USE_CONDVARS
Options, 35
CH_CFG_USE_DYNAMIC
Options, 38
CH_CFG_USE_EVENTS_TIMEOUT
Options, 36
CH_CFG_USE_EVENTS
Options, 36
CH_CFG_USE_FACTORY
Options, 38
CH_CFG_USE_HEAP
Options, 37
CH_CFG_USE_MAILBOXES
Options, 36
CH_CFG_USE_MEMCORE
Options, 37
CH_CFG_USE_MEMPOOLS
INDEX 473

CH_CFG_USE_MESSAGES_PRIORITY
  Options, 36
CH_CFG_USE_MESSAGES
  Options, 36
CH_CFG_USE_MUTEXES_RECURSIVE
  Options, 35
CH_CFG_USE_MUTEXES
  Options, 35
CH_CFG_USE_OBJ_FIFOS
  Options, 37
CH_CFG_USE_PIPES
  Options, 37
CH_CFG_USE_REGISTRY
  Options, 34
CH_CFG_USE_SEMAPHORES_PRIORITY
  Options, 35
CH_CFG_USE_SEMAPHORES
  Options, 35
CH_CFG_USE_TM
  Options, 34
CH_CFG_USE_WAITEXIT
  Options, 34
CH_CUSTOMER_ID_CODE
  Customer Information, 22
CH_CUSTOMER_ID_STRING
  Customer Information, 22
CH_DBG_ENABLE_ASSERTS
  Options, 39
CH_DBG_ENABLE_CHECKS
  Options, 39
CH_DBG_ENABLE_STACK_CHECK
  Options, 39
CH_DBG_FILL_THREADS
  Options, 40
CH_DBG_STACK_FILL_VALUE
  Checks and Assertions, 232
CH_DBG_STATISTICS
  Options, 39
CH_DBG_SYSTEM_STATE_CHECK
  Options, 39
CH_DBG_THREADS_PROFILING
  Options, 40
CH_DBG_TRACE_BUFFER_SIZE
  Tracing, 240
CH_DBG_TRACE_MASK
  Tracing, 240
CH_FAST_IRQ_HANDLER
  System Management, 52
CH_FLAG_MODE_HEAP
  Scheduler, 74
CH_FLAG_MODE_MASK
  Scheduler, 73
CH_FLAG_MODE_MPOOL
  Scheduler, 74
CH_FLAG_MODE_STATIC
  Scheduler, 74
CH_FLAG_TERMINATE
  Scheduler, 74
CH_HEAP_ALIGNMENT
  Memory Heaps, 295
CH_HEAP_AREA
  Memory Heaps, 295
CH_IRQ_EPILOGUE
  System Management, 52
CH_IRQ_HANDLER
  System Management, 52
CH_IRQ_IS_VALID_KERNEL_PRIORITY
  System Management, 51
CH_IRQ_IS_VALID_PRIORITY
  System Management, 51
CH_IRQ_PROLOGUE
  System Management, 51
CH_KERNEL_MAJOR
  Version Numbers and Identification, 27
CH_KERNEL_MINOR
  Version Numbers and Identification, 27
CH_KERNEL_PATCH
  Version Numbers and Identification, 27
CH_KERNEL_STABLE
  Version Numbers and Identification, 26
CH_KERNEL_VERSION
  Version Numbers and Identification, 26
CH_LICENSE_FEATURES
  License Settings, 24
CH_LICENSE_ID_CODE
  License Settings, 24
CH_LICENSE_ID_STRING
  License Settings, 23
CH_LICENSE_MAX_DEPLOY
  License Settings, 24
CH_LICENSE_MODIFIABLE_CODE
  License Settings, 24
CH_LICENSE_TYPE_STRING
  License Settings, 23
CH_LICENSE
  Customer Information, 22
CH_OSLIB_MAJOR
  Version Numbers and Identification, 251
CH_OSLIB_MINOR
  Version Numbers and Identification, 251
CH_OSLIB_PATCH
  Version Numbers and Identification, 251
CH_OSLIB_STABLE
  Version Numbers and Identification, 251
CH_OSLIB_VERSION
  Version Numbers and Identification, 251
CH_STATE_CURRENT
  Scheduler, 72
CH_STATE_FINAL
  Scheduler, 73
CH_STATE_NAMES
  Scheduler, 73
CH_STATE_QUEUED
  Scheduler, 72
CH_STATE_READY
Scheduler, 72
CH_STATE_SLEEPING
Scheduler, 73
CH_STATE SNDMSGQ
Scheduler, 73
CH_STATE SNDMSG
Scheduler, 73
CH_STATE SUSPENDED
Scheduler, 72
CH_STATE WTANDEVT
Scheduler, 73
CH_STATE WTCOND
Scheduler, 73
CH_STATE WTEXIT
Scheduler, 73
CH_STATE WTMSG
Scheduler, 73
CH_STATE WTMTX
Scheduler, 72
CH_STATE WTOREVT
Scheduler, 73
CH_STATE WTSEM
Scheduler, 72
CH_STATE WTSTART
Scheduler, 72
CH_VERSION_MONTH
Release Information, 20
CH_VERSION NICKNAME
Release Information, 21
CH_VERSION PATCH
Release Information, 20
CH_VERSION STABLE
Release Information, 20
CH_VERSION YEAR
Release Information, 20
CH_VERSION
Release Information, 20
CONDVAR DECL
Condition Variables, 184
ch
Scheduler, 89
ch.h, 417
ch_binary_semaphore, 351
ch_dyn_element, 352
next, 353
refs, 353
ch_dyn_list, 353
ch_dyn_mailbox, 354
element, 355
mbx, 355
msgbuf, 356
ch_dyn_object, 356
buffer, 357
element, 357
ch_dyn_objects_fifo, 357
element, 358
fifo, 358
msgbuf, 358
ch_dyn_pipe, 359
buffer, 361
element, 361
pipe, 361
chDyn_semaphore, 361
element, 362
sem, 362
ch factory
Dynamic Objects Factory, 350
ch_memcore
Core Memory Manager, 293
ch_mutex, 363
cnt, 364
next, 364
owner, 364
queue, 364
ch_objects_factory, 364
buf_list, 366
fifo_list, 366
mbx_list, 366
mtx, 366
obj_list, 366
obj_pool, 366
pipe_list, 366
sem_list, 366
sem_pool, 366
ch_objects_fifo, 367
free, 368
mbx, 368
ch_registered_static_object, 368
element, 369
objp, 369
ch_semaphore, 369
cnt, 370
queue, 370
ch system, 371
dbg, 372
kernel_stats, 372
mainthread, 372
rlist, 372
tm, 372
vtlist, 372
ch_system_debug, 372
isr_cnt, 374
lock_cnt, 374
panic_msg, 374
trace buffer, 374
ch_system_t
Scheduler, 75
ch_thread, 374
cctx, 377
epending, 379
ewmask, 379
exitcode, 378
flags, 377
mpool, 380
msgqueue, 379
INDEX 475

mtxlist, 379
name, 377
newer, 377
older, 377
prio, 377
queue, 377
rdmsg, 378
realprio, 380
refs, 377
sentmsg, 378
state, 377
stats, 380
ticks, 378
time, 378
u, 379
wabase, 377
waiting, 379
wtmtxp, 379
wtobjp, 378
wtsemp, 379
wttrp, 378
ch_threads_list, 380
next, 382
ch_threads_queue, 382
next, 383
prev, 383
ch_trace_buffer_t, 383
buffer, 385
ptr, 385
size, 385
suspended, 385
ch_trace_event_t, 385
halt, 388
isr, 388
name, 388
ntp, 388
reason, 388
rtstamp, 387
state, 387
sw, 388
time, 388
type, 387
up1, 388
up2, 388
user, 388
wtobjp, 388
ch_virtual_timer, 389
delta, 390
func, 391
next, 390
par, 391
prev, 390
ch_virtual_timers_list, 391
delta, 393
lasttime, 393
next, 392
prev, 392
systime, 393
chBSemGetStateI
Binary Semaphores, 261
chBSemObjectInit
Binary Semaphores, 254
chBSemReset
Binary Semaphores, 258
chBSemResetI
Binary Semaphores, 258
chBSemSignal
Binary Semaphores, 260
chBSemSignalI
Binary Semaphores, 259
chBSemWait
Binary Semaphores, 255
chBSemWaitTimeout
Binary Semaphores, 257
chBSemWaitTimeoutI
Binary Semaphores, 256
chBSemWaitS
Binary Semaphores, 255
chCondBroadcast
Condition Variables, 187
chCondBroadcastI
Condition Variables, 188
chCondObjectInit
Condition Variables, 184
chCondSignal
Condition Variables, 185
chCondSignalI
Condition Variables, 186
chCondWait
Condition Variables, 189
chCondWaitTimeout
Condition Variables, 191
chCondWaitTimeoutI
Condition Variables, 192
chCondWaitS
Condition Variables, 190
chCoreAlloc
Core Memory Manager, 292
chCoreAllocAligned
Core Memory Manager, 291
chCoreAllocAlignedWithOffset
Core Memory Manager, 289
chCoreAllocAlignedWithOffsetI
Core Memory Manager, 288
chCoreAllocAlignedI
Core Memory Manager, 290
chCoreAlloc
Core Memory Manager, 292
chCoreGetStatusX
Core Memory Manager, 290
chDbgAssert
Checks and Assertions, 233
chDbgCheck
Checks and Assertions, 232
chDbgCheckClassI
Checks and Assertions, 237
<table>
<thead>
<tr>
<th>Function</th>
<th>Category</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>chDbgCheckClassS</td>
<td>Checks and Assertions</td>
<td>237</td>
</tr>
<tr>
<td>chDbgResumeTrace</td>
<td>Tracing</td>
<td>245</td>
</tr>
<tr>
<td>chDbgResumeTraceI</td>
<td>Tracing</td>
<td>245</td>
</tr>
<tr>
<td>chDbgSuspendTrace</td>
<td>Tracing</td>
<td>244</td>
</tr>
<tr>
<td>chDbgSuspendTraceI</td>
<td>Tracing</td>
<td>244</td>
</tr>
<tr>
<td>chDbgWriteTrace</td>
<td>Tracing</td>
<td>243</td>
</tr>
<tr>
<td>chDbgWriteTraceI</td>
<td>Tracing</td>
<td>243</td>
</tr>
<tr>
<td>chEvtAddEvents</td>
<td>Event Flags</td>
<td>199</td>
</tr>
<tr>
<td>chEvtAddEventsI</td>
<td>Event Flags</td>
<td>213</td>
</tr>
<tr>
<td>chEvtBroadcast</td>
<td>Event Flags</td>
<td>212</td>
</tr>
<tr>
<td>chEvtBroadcastFlags</td>
<td>Event Flags</td>
<td>203</td>
</tr>
<tr>
<td>chEvtBroadcastFlagsI</td>
<td>Event Flags</td>
<td>200</td>
</tr>
<tr>
<td>chEvtBroadcastI</td>
<td>Event Flags</td>
<td>213</td>
</tr>
<tr>
<td>chEvtDispatch</td>
<td>Event Flags</td>
<td>204</td>
</tr>
<tr>
<td>chEvtGetAndClearEvents</td>
<td>Event Flags</td>
<td>198</td>
</tr>
<tr>
<td>chEvtGetAndClearEventsI</td>
<td>Event Flags</td>
<td>198</td>
</tr>
<tr>
<td>chEvtGetAndClearFlags</td>
<td>Event Flags</td>
<td>201</td>
</tr>
<tr>
<td>chEvtGetAndClearFlagsI</td>
<td>Event Flags</td>
<td>203</td>
</tr>
<tr>
<td>chEvtGetEventsX</td>
<td>Event Flags</td>
<td>214</td>
</tr>
<tr>
<td>chEvtIsListeningI</td>
<td>Event Flags</td>
<td>212</td>
</tr>
<tr>
<td>chEvtObjectInit</td>
<td>Event Flags</td>
<td>210</td>
</tr>
<tr>
<td>chEvtRegister</td>
<td>Event Flags</td>
<td>211</td>
</tr>
<tr>
<td>chEvtRegisterMask</td>
<td>Event Flags</td>
<td>211</td>
</tr>
<tr>
<td>chEvtRegisterMaskWithFlags</td>
<td>Event Flags</td>
<td>197</td>
</tr>
<tr>
<td>chEvtSignal</td>
<td>Event Flags</td>
<td>201</td>
</tr>
<tr>
<td>chEvtSignalAll</td>
<td>Event Flags</td>
<td>202</td>
</tr>
<tr>
<td>chEvtUnregister</td>
<td>Event Flags</td>
<td>197</td>
</tr>
<tr>
<td>chEvtWaitAll</td>
<td>Event Flags</td>
<td>206</td>
</tr>
<tr>
<td>chEvtWaitAllTimeout</td>
<td>Event Flags</td>
<td>209</td>
</tr>
<tr>
<td>chEvtWaitAny</td>
<td>Event Flags</td>
<td>205</td>
</tr>
<tr>
<td>chEvtWaitAnyTimeout</td>
<td>Event Flags</td>
<td>208</td>
</tr>
<tr>
<td>chEvtWaitOne</td>
<td>Event Flags</td>
<td>204</td>
</tr>
<tr>
<td>chEvtWaitOneTimeout</td>
<td>Event Flags</td>
<td>207</td>
</tr>
<tr>
<td>chFactoryCreateBuffer</td>
<td>Dynamic Objects Factory</td>
<td>340</td>
</tr>
<tr>
<td>chFactoryCreateMailbox</td>
<td>Dynamic Objects Factory</td>
<td>343</td>
</tr>
<tr>
<td>chFactoryCreateObjectsFIFO</td>
<td>Dynamic Objects Factory</td>
<td>344</td>
</tr>
<tr>
<td>chFactoryCreatePipe</td>
<td>Dynamic Objects Factory</td>
<td>346</td>
</tr>
<tr>
<td>chFactoryCreateSemaphore</td>
<td>Dynamic Objects Factory</td>
<td>341</td>
</tr>
<tr>
<td>chFactoryDuplicateReference</td>
<td>Dynamic Objects Factory</td>
<td>347</td>
</tr>
<tr>
<td>chFactoryFindBuffer</td>
<td>Dynamic Objects Factory</td>
<td>340</td>
</tr>
<tr>
<td>chFactoryFindMailbox</td>
<td>Dynamic Objects Factory</td>
<td>343</td>
</tr>
<tr>
<td>chFactoryFindObject</td>
<td>Dynamic Objects Factory</td>
<td>338</td>
</tr>
<tr>
<td>chFactoryFindObjectByPointer</td>
<td>Dynamic Objects Factory</td>
<td>339</td>
</tr>
<tr>
<td>chFactoryFindObjectFIFO</td>
<td>Dynamic Objects Factory</td>
<td>345</td>
</tr>
<tr>
<td>chFactoryFindPipe</td>
<td>Dynamic Objects Factory</td>
<td>346</td>
</tr>
<tr>
<td>chFactoryFindSemaphore</td>
<td>Dynamic Objects Factory</td>
<td>342</td>
</tr>
<tr>
<td>chFactoryGetBuffer</td>
<td>Dynamic Objects Factory</td>
<td>348</td>
</tr>
<tr>
<td>chFactoryGetBufferSize</td>
<td>Dynamic Objects Factory</td>
<td>348</td>
</tr>
<tr>
<td>chFactoryGetMailbox</td>
<td>Dynamic Objects Factory</td>
<td>349</td>
</tr>
<tr>
<td>chFactoryGetObject</td>
<td>Dynamic Objects Factory</td>
<td>348</td>
</tr>
<tr>
<td>chFactoryGetObjectsFIFO</td>
<td>Dynamic Objects Factory</td>
<td>349</td>
</tr>
<tr>
<td>chFactoryGetPipe</td>
<td>Dynamic Objects Factory</td>
<td>350</td>
</tr>
<tr>
<td>chFactoryGetSemaphore</td>
<td>Dynamic Objects Factory</td>
<td>349</td>
</tr>
<tr>
<td>chFactoryRegisterObject</td>
<td>Dynamic Objects Factory</td>
<td>338</td>
</tr>
<tr>
<td>chFactoryReleaseBuffer</td>
<td>Dynamic Objects Factory</td>
<td>341</td>
</tr>
<tr>
<td>chFactoryReleaseMailbox</td>
<td>Dynamic Objects Factory</td>
<td>344</td>
</tr>
<tr>
<td>chFactoryReleaseObject</td>
<td>Dynamic Objects Factory</td>
<td>339</td>
</tr>
<tr>
<td>chFactoryReleaseObjectsFIFO</td>
<td>Dynamic Objects Factory</td>
<td>345</td>
</tr>
</tbody>
</table>
chFactoryReleasePipe
  Dynamic Objects Factory, 347
chFactoryReleaseSemaphore
  Dynamic Objects Factory, 342
chFifoObjectInit
  Objects FIFOs, 321
chFifoObjectInitAligned
  Objects FIFOs, 321
chFifoReceiveObjectTimeout
  Objects FIFOs, 331
chFifoReceiveObjectTimeoutS
  Objects FIFOs, 330
chFifoReceiveObject
  Objects FIFOs, 330
chFifoReturnObject
  Objects FIFOs, 325
chFifoReturnObjectI
  Objects FIFOs, 324
chFifoSendObject
  Objects FIFOs, 327
chFifoSendObjectAhead
  Objects FIFOs, 329
chFifoSendObjectAheadI
  Objects FIFOs, 328
chFifoSendObjectAheadS
  Objects FIFOs, 328
chFifoSendObject
  Objects FIFOs, 326
chFifoSendObjectS
  Objects FIFOs, 326
chFifoTakeObjectTimeout
  Objects FIFOs, 323
chFifoTakeObjectTimeoutS
  Objects FIFOs, 323
chFifoTakeObject
  Objects FIFOs, 322
chGuardedPoolAdd
  Memory Pools, 315
chGuardedPoolAddI
  Memory Pools, 316
chGuardedPoolAddS
  Memory Pools, 317
chGuardedPoolAllocTimeout
  Memory Pools, 308
chGuardedPoolAllocTimeoutS
  Memory Pools, 308
chGuardedPoolAlloc
  Memory Pools, 313
chGuardedPoolFree
  Memory Pools, 309
chGuardedPoolFreeI
  Memory Pools, 314
chGuardedPoolFreeS
  Memory Pools, 315
chGuardedPoolGetCounterI
  Memory Pools, 313
chGuardedPoolLoadArray
  Memory Pools, 307
chGuardedPoolObjectInit
  Memory Pools, 312
chGuardedPoolObjectInitAligned
  Memory Pools, 306
chHeapAlloc
  Memory Heaps, 298
chHeapAllocAligned
  Memory Heaps, 296
chHeapFree
  Memory Heaps, 297
chHeapGetSize
  Memory Heaps, 298
chHeapObjectInit
  Memory Heaps, 296
chHeapStatus
  Memory Heaps, 297
chMBFetchTimeout
  Mailboxes, 272
chMBFetchTimeoutS
  Mailboxes, 273
chMBFetchI
  Mailboxes, 274
chMBGetFreeCountI
  Mailboxes, 276
chMBGetSizeI
  Mailboxes, 275
chMBGetUsedCountI
  Mailboxes, 275
chMBObjectInit
  Mailboxes, 265
chMBPeekI
  Mailboxes, 276
chMBPostAheadTimeout
  Mailboxes, 269
chMBPostAheadTimeoutS
  Mailboxes, 270
chMBPostAheadI
  Mailboxes, 271
chMBPostTimeout
  Mailboxes, 267
chMBPostTimeoutS
  Mailboxes, 268
chMBPostI
  Mailboxes, 269
chMBReset
  Mailboxes, 265
chMBResetI
  Mailboxes, 266
chMBResumeX
  Mailboxes, 277
chMsgGet
  Synchronous Messages, 218
chMsgIsPendingI
  Synchronous Messages, 218
chMsgRelease
  Synchronous Messages, 217
chMsgReleaseS
Synchronous Messages, 219
chMsgSend
Synchronous Messages, 215
chMsgWait
Synchronous Messages, 216
chMtxGetNextMutexX
Mutexes, 182
chMtxGetOwnerI
Mutexes, 181
chMtxLock
Mutexes, 173
chMtxLockS
Mutexes, 174
chMtxObjectInit
Mutexes, 173
chMtxQueueNotEmptyS
Mutexes, 180
chMtxTryLock
Mutexes, 175
chMtxTryLockS
Mutexes, 176
chMtxUnlock
Mutexes, 177
chMtxUnlockAll
Mutexes, 179
chMtxUnlockAllS
Mutexes, 179
chMtxUnlockS
Mutexes, 178
chPipeGetFreeCount
Pipes, 285
chPipeGetSize
Pipes, 284
chPipeGetUsedCount
Pipes, 284
chPipeObjectInit
Pipes, 280
chPipeReadTimeout
Pipes, 283
chPipeReset
Pipes, 281
chPipeResume
Pipes, 285
chPipeWriteTimeout
Pipes, 282
chPoolAdd
Memory Pools, 311
chPoolAddI
Memory Pools, 312
chPoolAlloc
Memory Pools, 304
chPoolAllocI
Memory Pools, 303
chPoolFree
Memory Pools, 306
chPoolFreeI
Memory Pools, 305
chPoolLoadArray
Memory Pools, 303
chPoolObjectInit
Memory Pools, 310
chPoolObjectInitAligned
Memory Pools, 302
chRegFindThreadByName
Registry, 226
chRegFindThreadByPointer
Registry, 227
chRegFindThreadByWorkingArea
Registry, 227
chRegFirstThread
Registry, 224
chRegGetNameX
Registry, 229
chRegNextThread
Registry, 225
chRegSetThreadName
Registry, 228
chRegSetThreadNameX
Registry, 229
chSchCanYieldS
Scheduler, 88
chSchDoReschedule
Scheduler, 84
chSchDoRescheduleAhead
Scheduler, 84
chSchDoRescheduleBehind
Scheduler, 83
chSchDoYieldS
Scheduler, 88
chSchGoSleepTimeoutS
Scheduler, 80
chSchGoSleepS
Scheduler, 80
chSchIsPreemptionRequired
Scheduler, 83
chSchIsRescRequiredI
Scheduler, 87
chSchPreemption
Scheduler, 89
chSchReadyAheadI
Scheduler, 79
chSchReadyI
Scheduler, 78
chSchRescheduleS
Scheduler, 82
chSchWakeupS
Scheduler, 81
chSemAddCounterI
Counting Semaphores, 166
chSemFastSignalI
Counting Semaphores, 169
chSemFastWaitI
Counting Semaphores, 168
chSemGetCounterI
Counting Semaphores, 169
INDEX

479

chSemObjectInit
  Counting Semaphores, 158
chSemReset
  Counting Semaphores, 158
chSemResetI
  Counting Semaphores, 159
chSemSignal
  Counting Semaphores, 164
chSemSignalWait
  Counting Semaphores, 167
chSemSignalI
  Counting Semaphores, 165
chSemWait
  Counting Semaphores, 160
chSemWaitTimeout
  Counting Semaphores, 162
chSemWaitTimeoutS
  Counting Semaphores, 163
chSysDisable
  System Management, 62
chSysEnable
  System Management, 63
chSysGetIdleThreadX
  System Management, 66
chSysGetRealtimeCounterX
  System Management, 55
chSysGetStatusAndLockX
  System Management, 59
chSysHalt
  System Management, 57
chSysInit
  Version Numbers and Identification, 27
chSysIntegrityCheckI
  System Management, 58
chSysIsCounterWithinX
  System Management, 61
chSysLock
  System Management, 63
chSysLockFromISR
  System Management, 64
chSysPolledDelayX
  System Management, 61
chSysRestoreStatusX
  System Management, 60
chSysSuspend
  System Management, 62
chSysSwitch
  System Management, 55
chSysTimerHandlerI
  System Management, 59
chSysUnconditionalLock
  System Management, 65
chSysUnconditionalUnlock
  System Management, 66
chSysUnlock
  System Management, 64
chSysUnlockFromISR
  System Management, 65
chTMChainMeasurementToX
  Time Measurement, 153
chTMOBJECTInit
  Time Measurement, 152
chTMStartMeasurementX
  Time Measurement, 153
chTMStopMeasurementX
  Time Measurement, 153
chThdAddRef
  Threads, 128
chThdCreate
  Threads, 125
chThdCreateFromHeap
  Dynamic Threads, 220
chThdCreateFromMemoryPool
  Dynamic Threads, 221
chThdCreateStatic
  Threads, 126
chThdCreateSuspended
  Threads, 123
chThdCreateSuspendedI
  Threads, 122
chThdCreateI
  Threads, 124
chThdDequeueAllI
  Threads, 144
chThdDequeueNextI
  Threads, 144
chThdDequeueNextI
  Threads, 150
chThdEnqueueTimeoutS
  Threads, 143
chThdExit
  Threads, 130
chThdExitS
  Threads, 131
chThdGetPriorityX
  Threads, 145
chThdGetSelfX
  Threads, 145
chThdGetTicksX
  Threads, 146
chThdGetWorkingAreaX
  Threads, 146
chThdQueueisEmptyI
  Threads, 149
chThdQueueObjectInit
  Threads, 148
chThdRelease
  Threads, 129
chThdResume
  Threads, 142
chThdResumeI
  Threads, 141
chThdResumeS
Threads, 141
chThdSetPriority
Threads, 133
chThdShouldTerminateX
Threads, 147
chThdSleep
Threads, 135
chThdSleepMicroseconds
Threads, 121
chThdSleepMilliseconds
Threads, 120
chThdSleepSeconds
Threads, 120
chThdSleepUntil
Threads, 136
chThdSleepUntilWindowed
Threads, 137
chThdSleepS
Threads, 148
chThdStart
Threads, 127
chThdStartI
Threads, 147
chThdSuspendTimeoutS
Threads, 140
chThdSuspendS
Threads, 139
chThdTerminate
Threads, 134
chThdTerminatedX
Threads, 146
chThdWait
Threads, 132
chThdYield
Threads, 138
chTimeAddX
Time and Intervals, 99
chTimeDiffX
Time and Intervals, 99
chTimeI2MS
Time and Intervals, 98
chTimeI2US
Time and Intervals, 98
chTimeI2S
Time and Intervals, 97
chTimeIsInRangeX
Time and Intervals, 99
chTimeMS2I
Time and Intervals, 97
chTimeS2I
Time and Intervals, 96
chTimeUS2I
Time and Intervals, 97
chVTDoResetI
Virtual Timers, 103
chVTDoSetI
Virtual Timers, 102
chVTDoTickI
Virtual Timers, 114
chVTGetSystemTime
Virtual Timers, 104
chVTGetSystemTimeX
Virtual Timers, 104
chVTGetTimersStateI
Virtual Timers, 107
chVTIsArmed
Virtual Timers, 109
chVTIsArmedI
Virtual Timers, 108
chVTIsSystemTimeWithin
Virtual Timers, 107
chVTIsSystemTimeWithinX
Virtual Timers, 106
chVTOBJECTInit
Virtual Timers, 104
chVRTReset
Virtual Timers, 111
chVRTResetI
Virtual Timers, 110
chVTS
Virtual Timers, 113
chVTSI
Virtual Timers, 112
chVTTimeElapsedSinceX
Virtual Timers, 105
chalign.h, 418
chbsem.h, 418
chchecks.h, 420
chcond.c, 420
chcond.h, 421
chconf.h, 422
chcustomer.h, 424
chdebug.c, 425
chdebug.h, 426
chdebug_t, 393
identifier, 394
off_ctx, 395
off_flags, 395
off_name, 395
off_newer, 395
off_older, 395
off_preempt, 396
off_prio, 395
off_refs, 395
off_state, 395
off_stklimit, 395
off_time, 396
ptrsize, 395
size, 394
threadsize, 395
timesize, 395
version, 394
zero, 394
chdynamic.c, 426
chdynamic.h, 426
Checks, 44
INDEX

Checks and Assertions, 231

_dbg_check_disable, 234
_dbg_check_enable, 234
_dbg_check_enter_isr, 236
_dbg_check_leave_isr, 237
_dbg_check_lock, 235
_dbg_check_lock_from_isr, 235
_dbg_check_suspend, 234
_dbg_check_unlock, 235
_dbg_check_unlock_from_isr, 236
CH_DBG_STACK_FILL_VALUE, 232
chDbgAssert, 233
chDbgCheck, 232
chDbgCheckClassI, 237
chDbgCheckClassS, 237
chevents.c, 427
chevents.h, 428
chfactory.c, 430
chfactory.h, 431
chlib.h, 434
chlicense.h, 435
chmboxes.c, 435
chmboxes.h, 436
chmemcore.c, 437
chmemcore.h, 438
chmemheaps.c, 439
chmemheaps.h, 439
chmempools.c, 440
chmempools.h, 441
chmsg.c, 443
chmsg.h, 443
chmtx.c, 444
chmtx.h, 444
chobjfifos.h, 445
chpipes.c, 447
chpipes.h, 447
chregistry.c, 448
chregistry.h, 449
chrestrictions.h, 450
chschd.c, 450
chschd.h, 451
chsem.c, 454
chsem.h, 455
chstats.c, 456
chstats.h, 456
chsystypes.c, 457
chsystypes.h, 457
chthreads.c, 460
chthreads.h, 461
chtime.h, 464
chtm.c, 465
chtm.h, 466
chtrace.c, 467
chtrace.h, 467
chversion.c, 469
chvt.c, 469
chvt.h, 469

cmtx

pipe_t, 411
cnt

ch_mutex, 364
ch_semaphore, 370
mailbox_t, 404
pipe_t, 410
Complex Services, 319
Condition Variables, 183

_CONDVAR_DATA, 184
CONDVAR_DECL, 184
chCondBroadcast, 187
chCondBroadcasttl, 188
chCondObjectInit, 184
chCondSignal, 185
chCondSignall, 186
chCondWait, 189
chCondWaitTimeout, 191
chCondWaitTimeoutS, 192
chCondWaitS, 190
condition_variable_t, 184
condition_variable, 396
queue, 397
condition_variable_t

Condition Variables, 184
Configuration, 29
Core Memory Manager, 287

_core_init, 288
CH_CFG_MEMCORE_SIZE, 288
ch_memcore, 293
chCoreAlloc, 292
chCoreAllocAligned, 291
chCoreAllocAlignedWithOffset, 289
chCoreAllocAlignedWithOffsettl, 288
chCoreAllocAlignedtl, 290
chCoreAlloc, 292
chCoreGetStatusX, 290
memgetfunc2_t, 288
memgetfunc_t, 288
Counting Semaphores, 156

_SEMAPHORE_DATA, 157
chSemAddCountertl, 166
chSemFastSignall, 169
chSemFastWait, 168
chSemGetCounterl, 169
chSemObjectInit, 158
chSemReset, 158
chSemGettl, 159
chSemSignal, 164
chSemSignalWait, 167
chSemSignall, 165
chSemWait, 160
chSemWaitTimeout, 162
chSemWaitTimeoutS, 163
chSemWaitS, 161
SEMAPHORE_DECL, 157
semaphore_t, 158
ctx
<table>
<thead>
<tr>
<th>ch_thread, 377</th>
<th>chFactoryGetObject, 348</th>
</tr>
</thead>
<tbody>
<tr>
<td>cumulative</td>
<td>chFactoryGetObjectsFIFO, 349</td>
</tr>
<tr>
<td>currp</td>
<td>chFactoryGetPipe, 350</td>
</tr>
<tr>
<td>Scheduler, 74</td>
<td>chFactoryGetSemaphore, 349</td>
</tr>
<tr>
<td>Customer Information, 22</td>
<td>chFactoryRegisterObject, 338</td>
</tr>
<tr>
<td>CH_CUSTOMER_ID_CODE, 22</td>
<td>chFactoryReleaseBuffer, 341</td>
</tr>
<tr>
<td>CH_CUSTOMER_ID_STRING, 22</td>
<td>chFactoryReleaseMailbox, 344</td>
</tr>
<tr>
<td>CH_LICENSE, 22</td>
<td>chFactoryReleaseObject, 339</td>
</tr>
<tr>
<td>dbg</td>
<td>chFactoryReleaseObjectsFIFO, 345</td>
</tr>
<tr>
<td>ch_system, 372</td>
<td>chFactoryReleasePipe, 347</td>
</tr>
<tr>
<td>Debug, 230</td>
<td>chFactoryReleaseSemaphore, 342</td>
</tr>
<tr>
<td>default_heap</td>
<td>dyn_buffer_t, 337</td>
</tr>
<tr>
<td>Memory Heaps, 299</td>
<td>dyn_element_t, 337</td>
</tr>
<tr>
<td>delta</td>
<td>dyn_list_t, 337</td>
</tr>
<tr>
<td>ch_virtual_timer, 390</td>
<td>dyn_mailbox_t, 337</td>
</tr>
<tr>
<td>ch_virtual_timers_list, 393</td>
<td>dyn_objects_fifo_t, 337</td>
</tr>
<tr>
<td>dyn_buffer_t</td>
<td>dyn_pipe_t, 337</td>
</tr>
<tr>
<td>Dynamic Objects Factory, 337</td>
<td>dyn_semaphore_t, 337</td>
</tr>
<tr>
<td>dyn_element_t</td>
<td>objects_factory_t, 337</td>
</tr>
<tr>
<td>Dynamic Objects Factory, 337</td>
<td>registered_object_t, 337</td>
</tr>
<tr>
<td>dyn_list_t</td>
<td>Dynamic Threads, 220</td>
</tr>
<tr>
<td>Dynamic Objects Factory, 337</td>
<td>chThdCreateFromHeap, 220</td>
</tr>
<tr>
<td>dyn_mailbox_t</td>
<td>chThdCreateFromMemoryPool, 221</td>
</tr>
<tr>
<td>Dynamic Objects Factory, 337</td>
<td>EVENT_MASK</td>
</tr>
<tr>
<td>dyn_objects_fifo_t</td>
<td>Event Flags, 196</td>
</tr>
<tr>
<td>Dynamic Objects Factory, 337</td>
<td>EVENTSOURCE_DECL</td>
</tr>
<tr>
<td>dyn_pipe_t</td>
<td>Event Flags, 196</td>
</tr>
<tr>
<td>Dynamic Objects Factory, 337</td>
<td>element</td>
</tr>
<tr>
<td>dyn_semaphore_t</td>
<td>ch_dyn_mailbox, 355</td>
</tr>
<tr>
<td>Dynamic Objects Factory, 337</td>
<td>ch_dyn_object, 357</td>
</tr>
<tr>
<td>Dynamic Objects Factory, 337</td>
<td>ch_dyn_objects_fifo, 358</td>
</tr>
<tr>
<td>Dynamic Objects Factory, 333</td>
<td>ch_dyn_pipe, 361</td>
</tr>
<tr>
<td>Dynamic Objects Factory, 333</td>
<td>ch_dyn_semaphore, 362</td>
</tr>
<tr>
<td>_factory_init, 338</td>
<td>ch_registered_static_object, 369</td>
</tr>
<tr>
<td>CH_CFG_FACTORY_GENERIC_BUFFERS, 336</td>
<td>endmem</td>
</tr>
<tr>
<td>CH_CFG_FACTORY_MAILBOXES, 336</td>
<td>memcore_t, 405</td>
</tr>
<tr>
<td>CH_CFG_FACTORY_MAX_NAMES_LENGTH, 336</td>
<td>epending</td>
</tr>
<tr>
<td>CH_CFG_FACTORY_OBJ_FIFOS, 336</td>
<td>ch_thread, 379</td>
</tr>
<tr>
<td>CH_CFG_FACTORY_OBJECTS_REGISTRY, 336</td>
<td>Event Flags, 194</td>
</tr>
<tr>
<td>CH_CFG_FACTORY_PIPES, 336, 337</td>
<td>EVENTSOURCE_DATA, 196</td>
</tr>
<tr>
<td>CH_CFG_FACTORY_SEMAPHORES, 336</td>
<td>ALL_EVENTS, 196</td>
</tr>
<tr>
<td>ch_factory, 350</td>
<td>chEvtAddEvents, 199</td>
</tr>
<tr>
<td>chFactoryCreateBuffer, 340</td>
<td>chEvtAddEventsI, 213</td>
</tr>
<tr>
<td>chFactoryCreateMailbox, 343</td>
<td>chEvtBroadcast, 212</td>
</tr>
<tr>
<td>chFactoryCreateObjectsFIFO, 344</td>
<td>chEvtBroadcastFlags, 203</td>
</tr>
<tr>
<td>chFactoryCreatePipe, 346</td>
<td>chEvtBroadcastFlagsI, 200</td>
</tr>
<tr>
<td>chFactoryCreateSemaphore, 341</td>
<td>chEvtBroadcastI, 213</td>
</tr>
<tr>
<td>chFactoryDuplicateReference, 347</td>
<td>chEvtDispatch, 204</td>
</tr>
<tr>
<td>chFactoryFindBuffer, 340</td>
<td>chEvtGetAndClearEvents, 198</td>
</tr>
<tr>
<td>chFactoryFindMailbox, 343</td>
<td>chEvtGetAndClearEventsI, 198</td>
</tr>
<tr>
<td>chFactoryFindObject, 338</td>
<td>chEvtGetAndClearFlags, 201</td>
</tr>
<tr>
<td>chFactoryFindObjectByPointer, 339</td>
<td>chEvtGetAndClearFlagsI, 203</td>
</tr>
<tr>
<td>chFactoryFindObjectFIFO, 345</td>
<td>chEvtGetEventsX, 214</td>
</tr>
<tr>
<td>chFactoryFindPipe, 346</td>
<td>chEvtIsListeningI, 212</td>
</tr>
<tr>
<td>chFactoryFindSemaphore, 342</td>
<td>chEvtObjectInit, 210</td>
</tr>
<tr>
<td>chFactoryGetBuffer, 348</td>
<td>chEvtRegister, 211</td>
</tr>
<tr>
<td>chFactoryGetBufferSize, 348</td>
<td>chEvtRegisterMask, 211</td>
</tr>
<tr>
<td>chFactoryGetMailbox, 349</td>
<td>chEvtRegisterMaskWithFlags, 197</td>
</tr>
</tbody>
</table>
MEMORYPOOL_DECL, 302
memory_heap, 406
header, 407
mtx, 407
provider, 407
memory_heap_t
Memory Heaps, 295
memory_pool_t, 407
align, 408
next, 408
object_size, 408
provider, 408
mpool
ch_thread, 380
msgbuf
ch_dyn_mailbox, 356
ch_dyn_objects_fifo, 358
msgqueue
ch_thread, 379
mtx
ch_objects_factory, 366
memory_heap, 407
mtxlist
ch_thread, 379
mutex_t
Mutexes, 173
Mutexes, 171
_MUTEX_DATA, 172
chMtxGetNextMutexX, 182
chMtxGetOwner1, 181
chMtxLock, 173
chMtxLockS, 174
chMtxObjectInit, 173
chMtxQueueNotEmptyS, 180
chMtxTryLock, 175
chMtxTryLockS, 176
chMtxUnlock, 177
chMtxUnlockAll, 179
chMtxUnlockAllS, 179
chMtxUnlockS, 178
MUTEX_DECL, 172
mutex_t, 173

n
time_measurement_t, 414

n_ctxswc
kernel_stats_t, 403

n_irq
kernel_stats_t, 403

NOPRIO
Scheduler, 72
NORMALPRIO
Scheduler, 72
name
ch_thread, 377
ch_trace_event_t, 388
thread_descriptor_t, 413
newer
ch_thread, 377

ch_dyn_element, 353
ch_mutex, 364
ch_threads_list, 382
ch_threads_queue, 383
ch_virtual_timer, 390
ch_virtual_timers_list, 392
event_listener, 398
event_source, 399
heap_header, 401
memory_pool_t, 408
pool_header, 412
nextmem
memcore_t, 405
ntp
ch_trace_event_t, 388

OS Library, 250
obj_list
ch_objects_factory, 366
obj_pool
ch_objects_factory, 366
object_size
memory_pool_t, 408

Objects FIFOs, 320
chFifoObjectInit, 321
chFifoObjectInitAligned, 321
chFifoReceiveObjectTimeout, 331
chFifoReceiveObjectTimeoutS, 330
chFifoReceiveObjectctl, 330
chFifoReturnObject, 325
chFifoReturnObjectctl, 324
chFifoReturnObjectS, 325
chFifoSendObject, 327
chFifoSendObjectAhead, 329
chFifoSendObjectAheadctl, 328
chFifoSendObjectAheadS, 328
chFifoSendObjectctl, 326
chFifoSendObjectS, 326
chFifoTakeObjectTimeout, 323
chFifoTakeObjectTimeoutS, 323
chFifoTakeObjectctl, 322
objects_fifo_t, 321

objects_factory_t
Dynamic Objects Factory, 337

objects_fifo_t
Objects FIFOs, 321

objp
ch_registered_static_object, 369

off_ctx
chdebug_t, 395

off_flags
chdebug_t, 395

off_name
chdebug_t, 395

off_newer
chdebug_t, 395

off_older
chdebug_t, 395
INDEX

off_preeempt  chdebug_t, 396
off_prio  chdebug_t, 395
off.refs  chdebug_t, 395
off_state  chdebug_t, 395
off_stklimit  chdebug_t, 395
off_time  chdebug_t, 396
offset  chdebug_t, 396

older  ch_thread, 377

Options, 30
CH_CFG_CONTEXT_SWITCH_HOOK, 41
CH_CFG_FACTORY_GENERIC_BUFFERS, 38
CH_CFG_FACTORY_MAILBOXES, 38
CH_CFG_FACTORY_MAX_NAMES_LENGTH, 38
CH_CFG_FACTORY_OBJ_FIFOS, 38
CH_CFG_FACTORY_OBJECTS_REGISTRY, 38
CH_CFG_FACTORY_PIPES, 39
CH_CFG_FACTORY_SEMAPHORES, 38
CH_CFG_IDLE_ENTER_HOOK, 42
CH_CFG_IDLE_LEAVE_HOOK, 42
CH_CFG_INTERVALS_SIZE, 33
CH_CFG_IRQ_EPILOGUE_HOOK, 41
CH_CFG_IRQ_PROLOGUE_HOOK, 41
CH_CFG_MEMCORE_SIZE, 33
CH_CFG_MEMORY, 34
CH_CFG_OPTIMIZE_SPEED, 34
CH_CFG_OPTIMIZE_SPEED, 34
CH_CFG_PIPES, 39
CH_CFG_SEMAPHORES_PRIORITY, 35
CH_CFG_SEMAPHORES, 35
CH_CFG_TM, 34
CH_CFG_USE_ASSERTS, 35
CH_CFG_USE_ASSERTS, 35
CH_CFG_USE_CONDVARS_TIMEOUT, 35
CH_CFG_USE_MUTEXES, 35
CH_CFG_USE_MUTEXES_RECURSIVE, 35
CH_CFG_USE_MUTEXES, 35
CH_DBG_ENABLE_ASSERTS, 39
CH_DBG_ENABLE_ASSERTS, 39
CH_DBG_ENABLE_CHECKS, 39
CH_DBG_ENABLE_CHECKS, 39
CH_DBG_ENABLE_STACK_CHECK, 39
CH_DBG_ENABLE_STACK_CHECK, 39
CH_DBG_ENABLE_TRACE_BUFFER_SIZE, 39
CH_DBG_ENABLE_TRACE_BUFFER_SIZE, 39

owner  ch_mutex, 364

PIECE DECL  Pipes, 279

pages  heap_header, 401

panic_msg  ch_system_debug, 374

par  ch_virtual_timer, 391

pipe  ch_dyn_pipe, 361

pipe_list  ch_objects_factory, 366
pipe_read  Pipes, 280

pipe_t, 408

buffer, 410
cmtx, 411
cnt, 410
rdptr, 410
reset, 410
rmtx, 411
rtr, 411
top, 410
twr, 411
wrptr, 410

pipe_write  Pipes, 279

Pipes, 278

_BEGIN_DATA, 278
chPipeGetFreeCount, 285
chPipeGetSize, 284
chPipeGetUsedCount, 284
chPipeObjectInit, 280
chPipeReadTimeout, 283
chPipeReset, 281
chPipeResume, 285