ChibiOS/RT

7.0.5

Reference Manual

Sat Sep 13 2025 08:11:26

1 Introduction 1
1.1 Kernel Concepts . . . 2
1.1.1 Naming Conventions . . . 2
1.1.2 API Name Suffixes . . . 2
1.1.3 Interrupt Classes . . . 3
1.1.4 System States . . . 3
1.1.5 Scheduling . . . 5
1.1.6 Thread States . . . 6
1.1.7 Priority Levels . . . 6
1.1.8 Thread Working Area . . . 6

2 Topic Index 9
2.1 Topics . . . 9

3 Hierarchical Index 11
3.1 Class Hierarchy . . . 11

4 Data Structure Index 13
4.1 Data Structures . . . 13

5 File Index 17
5.1 File List . . . 17

6 Topic Documentation 21
6.1 Compiler portability. . . . 21
6.1.1 Detailed Description . . . 21
6.1.2 Macro Definition Documentation . . . 22
6.1.2.1 CC_SECTION . . . 22
6.1.2.2 CC_WEAK . . . 22
6.1.2.3 CC_USED . . . 22
6.1.2.4 CC_ALIGN_DATA . . . 22
6.1.2.5 CC_ALIGN_CODE . . . 23
6.1.2.6 CC_PACK . . . 23
6.1.2.7 CC_NO_INLINE . . . 23
6.1.2.8 CC_FORCE_INLINE . . . 23
6.1.2.9 CC_NO_RETURN . . . 24
6.1.2.10 CC_ROMCONST . . . 24
6.1.2.11 CC_LIKELY . . . 24
6.1.2.12 CC_UNLIKELY . . . 24
6.2 Release and Licensing . . . 25
6.2.1 Detailed Description . . . 25
6.2.2 Release Information . . . 25
6.2.2.1 Detailed Description . . . 25
6.2.2.2 Macro Definition Documentation . . . 25

6.2.3 Customer Information . . . 27
6.2.3.1 Detailed Description . . . 27
6.2.3.2 Macro Definition Documentation . . . 27
6.2.4 License Settings . . . 30
6.2.4.1 Detailed Description . . . 30
6.2.4.2 Macro Definition Documentation . . . 31
6.3 RT Kernel . . . 33
6.3.1 Detailed Description . . . 33
6.3.2 Version Numbers and Identification . . . 34
6.3.2.1 Detailed Description . . . 34
6.3.2.2 Macro Definition Documentation . . . 34
6.3.3 Configuration . . . 36
6.3.3.1 Detailed Description . . . 36
6.3.3.2 Options . . . 36
6.3.3.3 Checks . . . 56
6.3.3.4 Restrictions . . . 56
6.3.4 System . . . 58
6.3.4.1 Detailed Description . . . 58
6.3.4.2 Port Interface . . . 58
6.3.4.3 OS Types and Structures . . . 59
6.3.4.4 OS Instances . . . 67
6.3.4.5 Runtime Faults Collection Unit . . . 70
6.3.4.6 Lists and Queues . . . 72
6.3.4.7 Scheduler . . . 84
6.3.5 Base Kernel Services . . . 103
6.3.5.1 Detailed Description . . . 103
6.3.5.2 System Management . . . 103
6.3.5.3 Memory Alignment . . . 126
6.3.5.4 Time and Intervals . . . 128
6.3.5.5 Virtual Timers . . . 141
6.3.5.6 Threads . . . 166
6.3.5.7 Time Measurement . . . 202
6.3.6 Synchronization . . . 206
6.3.6.1 Detailed Description . . . 206
6.3.6.2 Counting Semaphores . . . 206
6.3.6.3 Mutexes . . . 221
6.3.6.4 Condition Variables . . . 233
6.3.6.5 Event Flags . . . 243
6.3.6.6 Synchronous Messages . . . 266
6.3.7 Dynamic Threads . . . 277
6.3.7.1 Detailed Description . . . 277
6.3.7.2 Function Documentation . . . 277

6.3.8 Registry . . . 280
6.3.8.1 Detailed Description . . . 280
6.3.8.2 Macro Definition Documentation . . . 281
6.3.8.3 Function Documentation . . . 282
6.3.8.4 Variable Documentation . . . 288
6.3.9 Debug . . . 289
6.3.9.1 Detailed Description . . . 289
6.3.9.2 Checks and Assertions . . . 289
6.3.9.3 Tracing . . . 299
6.3.9.4 Statistics . . . 310
6.4 OS Library . . . 314
6.4.1 Detailed Description . . . 314
6.4.2 Version Numbers and Identification . . . 314
6.4.2.1 Detailed Description . . . 314
6.4.2.2 Macro Definition Documentation . . . 315
6.4.2.3 Function Documentation . . . 317
6.4.3 Synchronization . . . 317
6.4.3.1 Detailed Description . . . 317
6.4.3.2 Binary Semaphores . . . 318
6.4.3.3 Mailboxes . . . 329
6.4.3.4 Pipes . . . 344
6.4.3.5 Delegate Threads . . . 354
6.4.3.6 Jobs Queues . . . 363
6.4.4 Memory Management . . . 375
6.4.4.1 Detailed Description . . . 375
6.4.4.2 Core Memory Manager . . . 375
6.4.4.3 Memory Heaps . . . 384
6.4.4.4 Memory Pools . . . 392
6.4.5 Complex Services . . . 413
6.4.5.1 Detailed Description . . . 413
6.4.5.2 Objects FIFOs . . . 413
6.4.5.3 Objects Caches . . . 427
6.4.5.4 Dynamic Objects Factory . . . 439

7 Data Structure Documentation **469**

7.1 ch_binary_semaphore Struct Reference . . . 469
7.1.1 Detailed Description . . . 470
7.1.2 Field Documentation . . . 470
7.1.2.1 sem . . . 470
7.2 ch_delta_list Struct Reference . . . 471
7.2.1 Detailed Description . . . 471
7.2.2 Field Documentation . . . 471

7.2.2.1 next . . . 471
7.2.2.2 prev . . . 471
7.2.2.3 delta . . . 472
7.3 ch_dyn_element Struct Reference . . . 472
7.3.1 Detailed Description . . . 472
7.3.2 Field Documentation . . . 472
7.3.2.1 next . . . 472
7.3.2.2 refs . . . 473
7.3.2.3 name . . . 473
7.4 ch_dyn_list Struct Reference . . . 473
7.4.1 Detailed Description . . . 473
7.4.2 Field Documentation . . . 474
7.4.2.1 next . . . 474
7.5 ch_dyn_mailbox Struct Reference . . . 474
7.5.1 Detailed Description . . . 475
7.5.2 Field Documentation . . . 475
7.5.2.1 element . . . 475
7.5.2.2 mbx . . . 475
7.6 ch_dyn_object Struct Reference . . . 475
7.6.1 Detailed Description . . . 476
7.6.2 Field Documentation . . . 476
7.6.2.1 element . . . 476
7.7 ch_dyn_objects_fifo Struct Reference . . . 476
7.7.1 Detailed Description . . . 477
7.7.2 Field Documentation . . . 477
7.7.2.1 element . . . 477
7.7.2.2 fifo . . . 477
7.8 ch_dyn_pipe Struct Reference . . . 477
7.8.1 Detailed Description . . . 479
7.8.2 Field Documentation . . . 479
7.8.2.1 element . . . 479
7.8.2.2 pipe . . . 479
7.9 ch_dyn_semaphore Struct Reference . . . 479
7.9.1 Detailed Description . . . 480
7.9.2 Field Documentation . . . 480
7.9.2.1 element . . . 480
7.9.2.2 sem . . . 480
7.10 ch_job_descriptor Struct Reference . . . 480
7.10.1 Detailed Description . . . 481
7.10.2 Field Documentation . . . 481
7.10.2.1 jobfunc . . . 481
7.10.2.2 jobarg . . . 481

7.11 ch_jobs_queue Struct Reference . . . 481
7.11.1 Detailed Description . . . 482
7.11.2 Field Documentation . . . 482
7.11.2.1 free . . . 482
7.11.2.2 mbx . . . 483
7.12 ch_list Struct Reference . . . 483
7.12.1 Detailed Description . . . 483
7.12.2 Field Documentation . . . 483
7.12.2.1 next . . . 483
7.13 ch_mutex Struct Reference . . . 484
7.13.1 Detailed Description . . . 484
7.13.2 Field Documentation . . . 485
7.13.2.1 queue . . . 485
7.13.2.2 owner . . . 485
7.13.2.3 next . . . 485
7.13.2.4 cnt . . . 485
7.14 ch_objects_cache Struct Reference . . . 485
7.14.1 Detailed Description . . . 487
7.14.2 Field Documentation . . . 487
7.14.2.1 hashn . . . 487
7.14.2.2 hashp . . . 487
7.14.2.3 objn . . . 487
7.14.2.4 objsz . . . 487
7.14.2.5 objvp . . . 488
7.14.2.6 lru . . . 488
7.14.2.7 cache_sem . . . 488
7.14.2.8 lru_sem . . . 488
7.14.2.9 readf . . . 488
7.14.2.10 writef . . . 488
7.15 ch_objects_factory Struct Reference . . . 489
7.15.1 Detailed Description . . . 490
7.15.2 Field Documentation . . . 490
7.15.2.1 mtx . . . 490
7.15.2.2 obj_list . . . 490
7.15.2.3 obj_pool . . . 490
7.15.2.4 buf_list . . . 490
7.15.2.5 sem_list . . . 490
7.15.2.6 sem_pool . . . 491
7.15.2.7 mbx_list . . . 491
7.15.2.8 fifo_list . . . 491
7.15.2.9 pipe_list . . . 491
7.16 ch_objects_fifo Struct Reference . . . 491

7.16.1 Detailed Description . . . 492
7.16.2 Field Documentation . . . 492
7.16.2.1 free . . . 492
7.16.2.2 mbx . . . 493
7.17 ch_oc_hash_header Struct Reference . . . 493
7.17.1 Detailed Description . . . 494
7.17.2 Field Documentation . . . 494
7.17.2.1 hash_next . . . 494
7.17.2.2 hash_prev . . . 494
7.18 ch_oc_lru_header Struct Reference . . . 494
7.18.1 Detailed Description . . . 495
7.18.2 Field Documentation . . . 496
7.18.2.1 hash_next . . . 496
7.18.2.2 hash_prev . . . 496
7.18.2.3 lru_next . . . 496
7.18.2.4 lru_prev . . . 496
7.19 ch_oc_object Struct Reference . . . 496
7.19.1 Detailed Description . . . 498
7.19.2 Field Documentation . . . 498
7.19.2.1 hash_next . . . 498
7.19.2.2 hash_prev . . . 498
7.19.2.3 lru_next . . . 498
7.19.2.4 lru_prev . . . 498
7.19.2.5 obj_group . . . 498
7.19.2.6 obj_key . . . 498
7.19.2.7 obj_sem . . . 499
7.19.2.8 obj_flags . . . 499
7.19.2.9 dptr . . . 499
7.20 ch_os_instance Struct Reference . . . 499
7.20.1 Detailed Description . . . 501
7.20.2 Field Documentation . . . 501
7.20.2.1 rlist . . . 501
7.20.2.2 vtlist . . . 501
7.20.2.3 reglist . . . 501
7.20.2.4 core_id . . . 501
7.20.2.5 rfcu . . . 502
7.20.2.6 config . . . 502
7.20.2.7 mainthread . . . 502
7.20.2.8 dbg . . . 502
7.20.2.9 trace_buffer . . . 502
7.20.2.10 kernel_stats . . . 502
7.21 ch_os_instance_config Struct Reference . . . 503

7.21.1 Detailed Description . . . 503
7.21.2 Field Documentation . . . 503
7.21.2.1 name . . . 503
7.21.2.2 mainthread_base . . . 504
7.21.2.3 mainthread_end . . . 504
7.21.2.4 idlethread_base . . . 504
7.21.2.5 idlethread_end . . . 504
7.22 ch_priority_queue Struct Reference . . . 504
7.22.1 Detailed Description . . . 505
7.22.2 Field Documentation . . . 505
7.22.2.1 next . . . 505
7.22.2.2 prev . . . 505
7.22.2.3 prio . . . 505
7.23 ch_queue Struct Reference . . . 505
7.23.1 Detailed Description . . . 506
7.23.2 Field Documentation . . . 506
7.23.2.1 next . . . 506
7.23.2.2 prev . . . 506
7.24 ch_ready_list Struct Reference . . . 506
7.24.1 Detailed Description . . . 507
7.24.2 Field Documentation . . . 507
7.24.2.1 pqueue . . . 507
7.24.2.2 current . . . 508
7.25 ch_registered_static_object Struct Reference . . . 508
7.25.1 Detailed Description . . . 508
7.25.2 Field Documentation . . . 509
7.25.2.1 element . . . 509
7.25.2.2 objp . . . 509
7.26 ch_registry Struct Reference . . . 509
7.26.1 Detailed Description . . . 510
7.26.2 Field Documentation . . . 510
7.26.2.1 queue . . . 510
7.27 ch_rfcu Struct Reference . . . 510
7.27.1 Detailed Description . . . 510
7.27.2 Field Documentation . . . 510
7.27.2.1 mask . . . 510
7.28 ch_semaphore Struct Reference . . . 511
7.28.1 Detailed Description . . . 512
7.28.2 Field Documentation . . . 512
7.28.2.1 queue . . . 512
7.28.2.2 cnt . . . 512
7.29 ch_system Struct Reference . . . 512

7.29.1 Detailed Description . . . 513
7.29.2 Field Documentation . . . 513
7.29.2.1 state . . . 513
7.29.2.2 instances . . . 513
7.29.2.3 tmc . . . 513
7.29.2.4 reglist . . . 513
7.29.2.5 rfcu . . . 514
7.30 ch_system_debug Struct Reference . . . 514
7.30.1 Detailed Description . . . 514
7.30.2 Field Documentation . . . 515
7.30.2.1 panic_msg . . . 515
7.30.2.2 isr_cnt . . . 515
7.30.2.3 lock_cnt . . . 515
7.31 ch_thread Struct Reference . . . 515
7.31.1 Detailed Description . . . 517
7.31.2 Field Documentation . . . 517
7.31.2.1 list . . . 517
7.31.2.2 queue . . . 517
7.31.2.3 pqueue . . . 517
7.31.2.4 [union] . . . 517
7.31.2.5 ctx . . . 518
7.31.2.6 rqueue . . . 518
7.31.2.7 owner . . . 518
7.31.2.8 name . . . 518
7.31.2.9 wabase . . . 518
7.31.2.10 state . . . 518
7.31.2.11 flags . . . 518
7.31.2.12 refs . . . 519
7.31.2.13 ticks . . . 519
7.31.2.14 time . . . 519
7.31.2.15 rdymsg . . . 519
7.31.2.16 exitcode . . . 519
7.31.2.17 wtobjp . . . 520
7.31.2.18 wttrp . . . 520
7.31.2.19 sentmsg . . . 520
7.31.2.20 wtsemp . . . 520
7.31.2.21 wtmtxp . . . 520
7.31.2.22 ewmask . . . 521
7.31.2.23 [union] . . . 521
7.31.2.24 waiting . . . 521
7.31.2.25 msgqueue . . . 521
7.31.2.26 epending . . . 521

7.31.2.27 mtxlist . . . 521
7.31.2.28 realprio . . . 522
7.31.2.29 mpool . . . 522
7.31.2.30 stats . . . 522
7.32 ch_threads_queue Struct Reference . . . 522
7.32.1 Detailed Description . . . 523
7.32.2 Field Documentation . . . 523
7.32.2.1 queue . . . 523
7.33 ch_virtual_timer Struct Reference . . . 523
7.33.1 Detailed Description . . . 524
7.33.2 Field Documentation . . . 524
7.33.2.1 dlist . . . 524
7.33.2.2 func . . . 524
7.33.2.3 par . . . 524
7.33.2.4 reload . . . 524
7.34 ch_virtual_timers_list Struct Reference . . . 524
7.34.1 Detailed Description . . . 525
7.34.2 Field Documentation . . . 525
7.34.2.1 dlist . . . 525
7.34.2.2 systime . . . 526
7.34.2.3 lasttime . . . 526
7.34.2.4 laststamp . . . 526
7.35 chdebug_t Struct Reference . . . 526
7.35.1 Detailed Description . . . 528
7.35.2 Field Documentation . . . 528
7.35.2.1 identifier . . . 528
7.35.2.2 zero . . . 528
7.35.2.3 size . . . 528
7.35.2.4 version . . . 528
7.35.2.5 ptrsize . . . 529
7.35.2.6 timesize . . . 529
7.35.2.7 threadsize . . . 529
7.35.2.8 off_prio . . . 529
7.35.2.9 off_ctx . . . 529
7.35.2.10 off_newer . . . 529
7.35.2.11 off_older . . . 529
7.35.2.12 off_name . . . 529
7.35.2.13 off_stklimit . . . 530
7.35.2.14 off_state . . . 530
7.35.2.15 off_flags . . . 530
7.35.2.16 off_refs . . . 530
7.35.2.17 off_preempt . . . 530

7.35.2.18 off_time . . . 530
7.35.2.19 off_reserved . . . 530
7.35.2.20 intctxsize . . . 530
7.35.2.21 intervalsize . . . 531
7.35.2.22 instancesnum . . . 531
7.35.2.23 off_sys_state . . . 531
7.35.2.24 off_sys_instances . . . 531
7.35.2.25 off_sys_reglist . . . 531
7.35.2.26 off_sys_rfcu . . . 531
7.35.2.27 off_sys_reserved . . . 531
7.35.2.28 off_inst_rlist_current . . . 531
7.35.2.29 off_inst_rlist . . . 532
7.35.2.30 off_inst_vtlist . . . 532
7.35.2.31 off_inst_reglist . . . 532
7.35.2.32 off_inst_core_id . . . 532
7.35.2.33 off_inst_rfcu . . . 532
7.36 condition_variable Struct Reference . . . 532
7.36.1 Detailed Description . . . 533
7.36.2 Field Documentation . . . 533
7.36.2.1 queue . . . 533
7.37 event_listener Struct Reference . . . 533
7.37.1 Detailed Description . . . 534
7.37.2 Field Documentation . . . 534
7.37.2.1 next . . . 534
7.37.2.2 listener . . . 534
7.37.2.3 events . . . 534
7.37.2.4 flags . . . 534
7.37.2.5 wflags . . . 535
7.38 event_source Struct Reference . . . 535
7.38.1 Detailed Description . . . 536
7.38.2 Field Documentation . . . 536
7.38.2.1 next . . . 536
7.39 guarded_memory_pool_t Struct Reference . . . 536
7.39.1 Detailed Description . . . 537
7.39.2 Field Documentation . . . 537
7.39.2.1 sem . . . 537
7.39.2.2 pool . . . 537
7.40 heap_header Union Reference . . . 537
7.40.1 Detailed Description . . . 538
7.40.2 Field Documentation . . . 538
7.40.2.1 next . . . 538
7.40.2.2 pages . . . 538

7.40.2.3 [struct] . . . 538
7.40.2.4 heap . . . 538
7.40.2.5 size . . . 539
7.40.2.6 [struct] . . . 539
7.41 kernel_stats_t Struct Reference . . . 539
7.41.1 Detailed Description . . . 540
7.41.2 Field Documentation . . . 540
7.41.2.1 n_irq . . . 540
7.41.2.2 n_ctxswc . . . 540
7.41.2.3 m_crit_thd . . . 540
7.41.2.4 m_crit_isr . . . 540
7.42 mailbox_t Struct Reference . . . 540
7.42.1 Detailed Description . . . 542
7.42.2 Field Documentation . . . 542
7.42.2.1 buffer . . . 542
7.42.2.2 top . . . 542
7.42.2.3 wrptr . . . 542
7.42.2.4 rdptr . . . 542
7.42.2.5 cnt . . . 542
7.42.2.6 reset . . . 542
7.42.2.7 qw . . . 543
7.42.2.8 qr . . . 543
7.43 memcore_t Struct Reference . . . 543
7.43.1 Detailed Description . . . 543
7.43.2 Field Documentation . . . 543
7.43.2.1 basemem . . . 543
7.43.2.2 topmem . . . 544
7.44 memory_heap Struct Reference . . . 544
7.44.1 Detailed Description . . . 546
7.44.2 Field Documentation . . . 546
7.44.2.1 provider . . . 546
7.44.2.2 header . . . 546
7.44.2.3 mtx . . . 546
7.45 memory_pool_t Struct Reference . . . 546
7.45.1 Detailed Description . . . 547
7.45.2 Field Documentation . . . 547
7.45.2.1 next . . . 547
7.45.2.2 object_size . . . 547
7.45.2.3 align . . . 547
7.45.2.4 provider . . . 547
7.46 pipe_t Struct Reference . . . 548
7.46.1 Detailed Description . . . 549

7.46.2 Field Documentation . . . 549
7.46.2.1 buffer . . . 549
7.46.2.2 top . . . 549
7.46.2.3 wrptr . . . 549
7.46.2.4 rdptr . . . 549
7.46.2.5 cnt . . . 550
7.46.2.6 reset . . . 550
7.46.2.7 wtr . . . 550
7.46.2.8 rtr . . . 550
7.46.2.9 cmtx . . . 550
7.46.2.10 wmtx . . . 550
7.46.2.11 rmtx . . . 550
7.47 pool_header Struct Reference . . . 551
7.47.1 Detailed Description . . . 551
7.47.2 Field Documentation . . . 551
7.47.2.1 next . . . 551
7.48 thread_descriptor_t Struct Reference . . . 551
7.48.1 Detailed Description . . . 552
7.48.2 Field Documentation . . . 553
7.48.2.1 name . . . 553
7.48.2.2 wbase . . . 553
7.48.2.3 wend . . . 553
7.48.2.4 prio . . . 553
7.48.2.5 funcp . . . 553
7.48.2.6 arg . . . 553
7.48.2.7 instance . . . 553
7.49 time_measurement_t Struct Reference . . . 554
7.49.1 Detailed Description . . . 554
7.49.2 Field Documentation . . . 555
7.49.2.1 best . . . 555
7.49.2.2 worst . . . 555
7.49.2.3 last . . . 555
7.49.2.4 n . . . 555
7.49.2.5 cumulative . . . 555
7.50 tm_calibration_t Struct Reference . . . 555
7.50.1 Detailed Description . . . 556
7.50.2 Field Documentation . . . 556
7.50.2.1 offset . . . 556
7.51 trace_buffer_t Struct Reference . . . 556
7.51.1 Detailed Description . . . 558
7.51.2 Field Documentation . . . 558
7.51.2.1 suspended . . . 558

7.51.2.2 size . . . 558
7.51.2.3 ptr . . . 558
7.51.2.4 buffer . . . 558
7.52 trace_event_t Struct Reference . . . 559
7.52.1 Detailed Description . . . 560
7.52.2 Field Documentation . . . 560
7.52.2.1 type . . . 560
7.52.2.2 state . . . 560
7.52.2.3 rtstamp . . . 561
7.52.2.4 time . . . 561
7.52.2.5 ntp . . . 561
7.52.2.6 wtobjp . . . 561
7.52.2.7 [struct] . . . 561
7.52.2.8 tp . . . 561
7.52.2.9 msg . . . 561
7.52.2.10 [struct] . . . 562
7.52.2.11 name . . . 562
7.52.2.12 [struct] . . . 562
7.52.2.13 reason . . . 562
7.52.2.14 [struct] . . . 562
7.52.2.15 up1 . . . 562
7.52.2.16 up2 . . . 562
7.52.2.17 [struct] . . . 562
7.52.2.18 [union] . . . 562

8 File Documentation **563**
8.1 concepts.dox File Reference . . . 563
8.2 main.dox File Reference . . . 563
8.3 ccportab.h File Reference . . . 563
8.3.1 Detailed Description . . . 564
8.4 chcustomer.h File Reference . . . 564
8.4.1 Detailed Description . . . 564
8.5 chlicense.h File Reference . . . 565
8.5.1 Detailed Description . . . 565
8.6 chversion.h File Reference . . . 566
8.6.1 Detailed Description . . . 566
8.7 license.dox File Reference . . . 566
8.8 lib.dox File Reference . . . 566
8.9 chbsem.h File Reference . . . 566
8.9.1 Detailed Description . . . 567
8.10 chdelegates.h File Reference . . . 568
8.10.1 Detailed Description . . . 569

8.11 chfactory.h File Reference . . . 569
8.11.1 Detailed Description . . . 572
8.12 chjobs.h File Reference . . . 572
8.12.1 Detailed Description . . . 573
8.13 chlib.h File Reference . . . 574
8.13.1 Detailed Description . . . 575
8.14 chmboxes.h File Reference . . . 575
8.14.1 Detailed Description . . . 576
8.15 chmemcore.h File Reference . . . 576
8.15.1 Detailed Description . . . 577
8.16 chmemheaps.h File Reference . . . 577
8.16.1 Detailed Description . . . 578
8.17 chmempools.h File Reference . . . 578
8.17.1 Detailed Description . . . 580
8.18 chobjcaches.h File Reference . . . 580
8.18.1 Detailed Description . . . 581
8.19 chobjfifos.h File Reference . . . 581
8.19.1 Detailed Description . . . 582
8.20 chpipes.h File Reference . . . 582
8.20.1 Detailed Description . . . 583
8.21 chdelegates.c File Reference . . . 583
8.21.1 Detailed Description . . . 584
8.22 chfactory.c File Reference . . . 584
8.22.1 Detailed Description . . . 586
8.23 chmboxes.c File Reference . . . 586
8.23.1 Detailed Description . . . 587
8.24 chmemcore.c File Reference . . . 587
8.24.1 Detailed Description . . . 587
8.25 chmemheaps.c File Reference . . . 587
8.25.1 Detailed Description . . . 588
8.26 chmempools.c File Reference . . . 588
8.26.1 Detailed Description . . . 589
8.27 chobjcaches.c File Reference . . . 589
8.27.1 Detailed Description . . . 590
8.28 chpipes.c File Reference . . . 591
8.28.1 Detailed Description . . . 591
8.29 rt.dox File Reference . . . 592
8.30 rt_arch.dox File Reference . . . 592
8.31 ch.h File Reference . . . 592
8.31.1 Detailed Description . . . 593
8.32 chalign.h File Reference . . . 593
8.32.1 Detailed Description . . . 593

8.33 chchecks.h File Reference . . . 594
8.33.1 Detailed Description . . . 594
8.34 chcond.h File Reference . . . 594
8.34.1 Detailed Description . . . 595
8.35 chdebug.h File Reference . . . 595
8.35.1 Detailed Description . . . 596
8.36 chdynamic.h File Reference . . . 596
8.36.1 Detailed Description . . . 596
8.37 chearly.h File Reference . . . 596
8.37.1 Detailed Description . . . 597
8.38 chevents.h File Reference . . . 597
8.38.1 Detailed Description . . . 599
8.39 chinstances.h File Reference . . . 599
8.39.1 Detailed Description . . . 600
8.40 chlists.h File Reference . . . 600
8.40.1 Detailed Description . . . 602
8.41 chmsg.h File Reference . . . 602
8.41.1 Detailed Description . . . 602
8.42 chmtx.h File Reference . . . 602
8.42.1 Detailed Description . . . 603
8.43 chobjects.h File Reference . . . 604
8.43.1 Detailed Description . . . 605
8.44 chport.h File Reference . . . 605
8.44.1 Detailed Description . . . 605
8.45 chregistry.h File Reference . . . 605
8.45.1 Detailed Description . . . 606
8.46 chrestrictions.h File Reference . . . 606
8.46.1 Detailed Description . . . 606
8.47 chrfcu.h File Reference . . . 607
8.47.1 Detailed Description . . . 607
8.48 chschd.h File Reference . . . 607
8.48.1 Detailed Description . . . 609
8.49 chsem.h File Reference . . . 610
8.49.1 Detailed Description . . . 611
8.50 chstats.h File Reference . . . 611
8.50.1 Detailed Description . . . 611
8.51 chsys.h File Reference . . . 611
8.51.1 Detailed Description . . . 613
8.52 chthreads.h File Reference . . . 613
8.52.1 Detailed Description . . . 616
8.53 chtime.h File Reference . . . 616
8.53.1 Detailed Description . . . 618

8.54 chtm.h File Reference . . . 618
8.54.1 Detailed Description . . . 618
8.55 chtrace.h File Reference . . . 618
8.55.1 Detailed Description . . . 620
8.56 chvt.h File Reference . . . 620
8.56.1 Detailed Description . . . 621
8.57 chcond.c File Reference . . . 622
8.57.1 Detailed Description . . . 622
8.58 chdebug.c File Reference . . . 622
8.58.1 Detailed Description . . . 623
8.59 chdynamic.c File Reference . . . 623
8.59.1 Detailed Description . . . 623
8.60 chevents.c File Reference . . . 624
8.60.1 Detailed Description . . . 625
8.61 chinstances.c File Reference . . . 625
8.61.1 Detailed Description . . . 625
8.62 chmsg.c File Reference . . . 625
8.62.1 Detailed Description . . . 625
8.63 chmtx.c File Reference . . . 626
8.63.1 Detailed Description . . . 626
8.64 chregistry.c File Reference . . . 626
8.64.1 Detailed Description . . . 627
8.65 chrfcu.c File Reference . . . 627
8.65.1 Detailed Description . . . 627
8.66 chschd.c File Reference . . . 627
8.66.1 Detailed Description . . . 628
8.67 chsem.c File Reference . . . 628
8.67.1 Detailed Description . . . 629
8.68 chstats.c File Reference . . . 629
8.68.1 Detailed Description . . . 630
8.69 chsys.c File Reference . . . 630
8.69.1 Detailed Description . . . 631
8.70 chthreads.c File Reference . . . 631
8.70.1 Detailed Description . . . 632
8.71 chtm.c File Reference . . . 632
8.71.1 Detailed Description . . . 633
8.72 chtrace.c File Reference . . . 633
8.72.1 Detailed Description . . . 634
8.73 chvt.c File Reference . . . 634
8.73.1 Detailed Description . . . 634
8.74 chconf.h File Reference . . . 634
8.74.1 Detailed Description . . . 637

Index 639

Chapter 1

Introduction

Author

Giovanni Di Sirio (gdisirio@users.sourceforge.net).

Chibi ?

I didn't want a serious name for this project. It is the Japanese word for small as in small child. So ChibiOS/RT means small Real Time Operating System. Source Wikipedia.

Features

- Free software, GPL3 licensed. Stable releases include a exception clause to the GPL.
- Designed for realtime applications.
- Easily portable.
- Preemptive scheduling.
- 128 priority levels. Multiple threads at the same priority level allowed.
- Round robin scheduling for threads at the same priority level.
- Offers threads, virtual timers, binary and counter semaphores, mutexes, condvars, event sources, event flags, messages, mailboxes.
- No static setup at compile time, there is no need to configure a maximum number of all the above objects.
- PC simulator target included, the development can be done on a PC under Linux or Windows.
  Timers, I/O channels and other HW resources are simulated in a guest OS process and the application code does not need to be aware of it.
- No *need* for a memory allocator, all the kernel structures are static and declaratively allocated.
- Optional, thread safe, Heap Allocator subsystem.
- Optional, thread safe, Memory Pools Allocator subsystem.
- Minimal system requirements: about 6KiB ROM with all options enabled and speed optimizations on. The size can shrink under 2KiB by disabling the the unused subsystems and optimizing for size.
- Almost totally written in C with little ASM code required for ports.
- Compatible with ChibiOS/HAL.

Related pages

- Kernel Concepts

1.1 Kernel Concepts

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

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

1.1.1 Naming Conventions

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

1.1.2 API Name Suffixes

The suffix can be one of the following:

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

Examples: `chThdCreateStatic()`, `chSemSignalI()`, `chThdCreateStatic()`.

1.1.3 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.

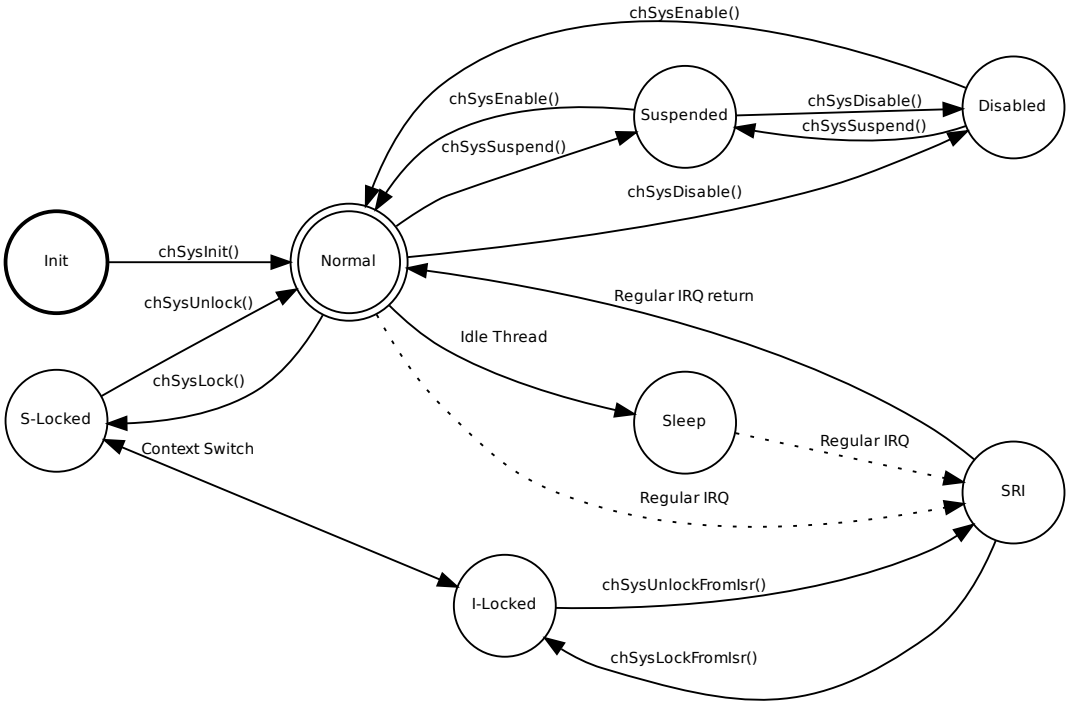
The mapping of the above logical classes into physical interrupts priorities is, of course, port dependent. See the documentation of the various ports for details.

1.1.4 System States

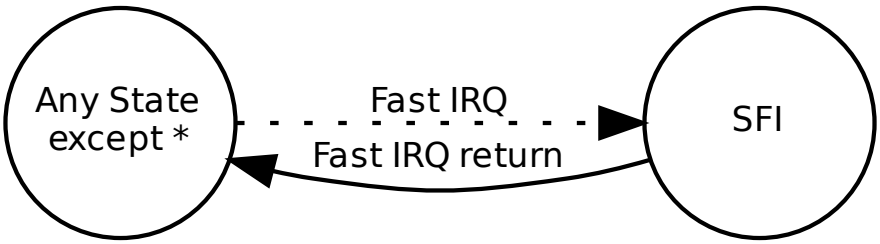
When using ChibiOS/RT the system can be in one of the following logical operating states:

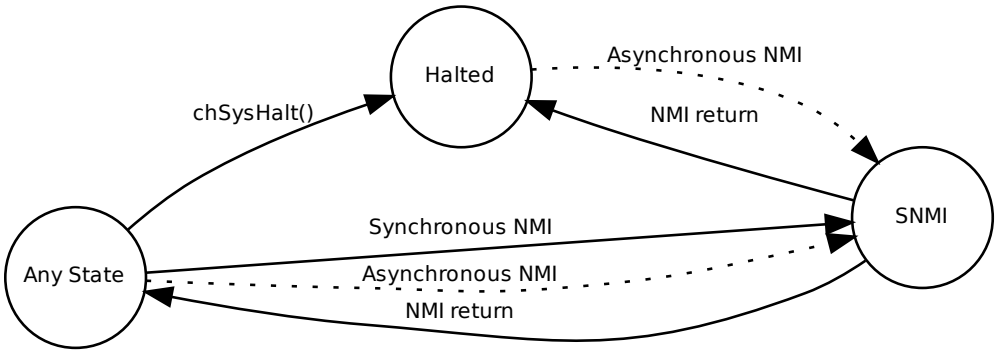
- **Init**. When the system is in this state all the maskable interrupt sources are disabled. In this state it is not possible to use any system API except `chSysInit()`. This state is entered after a physical reset.
- **Normal**. All the interrupt sources are enabled and the system APIs are accessible, threads are running.
- **Suspended**. In this state the fast interrupt sources are enabled but the regular interrupt sources are not. In this state it is not possible to use any system API except `chSysDisable()` or `chSysEnable()` in order to change state.
- **Disabled**. When the system is in this state both the maskable regular and fast interrupt sources are disabled. In this state it is not possible to use any system API except `chSysSuspend()` or `chSysEnable()` in order to change state.
- **Sleep**. Architecture-dependent low power mode, the idle thread goes in this state and waits for interrupts, after servicing the interrupt the Normal state is restored and the scheduler has a chance to reschedule.
- **S-Locked**. Kernel locked and regular interrupt sources disabled. Fast interrupt sources are enabled. S-Class and I-Class APIs are invokable in this state.
- **I-Locked**. Kernel locked and regular interrupt sources disabled. I-Class APIs are invokable from this state.
- **Serving Regular Interrupt**. No system APIs are accessible but it is possible to switch to the I-Locked state using `chSysLockFromIsr()` and then invoke any I-Class API. Interrupt handlers can be preemptable on some architectures thus is important to switch to I-Locked state before invoking system APIs.
- **Serving Fast Interrupt**. System APIs are not accessible.
- **Serving Non-Maskable Interrupt**. System APIs are not accessible.
- **Halted**. All interrupt sources are disabled and system stopped into an infinite loop. This state can be reached if the debug mode is activated **and** an error is detected **or** after explicitly invoking `chSysHalt()`.

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



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





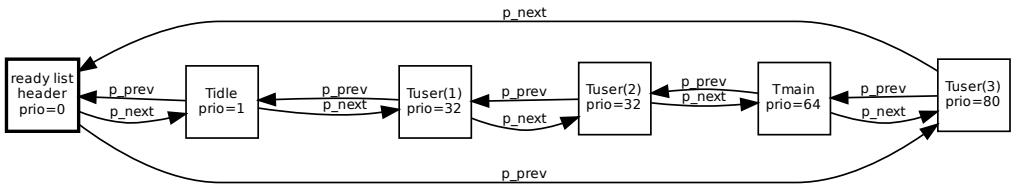
Attention

* except: **Init**, **Halt**, **SNMI**, **Disabled**.

1.1.5 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.

Threads Ready for Execution

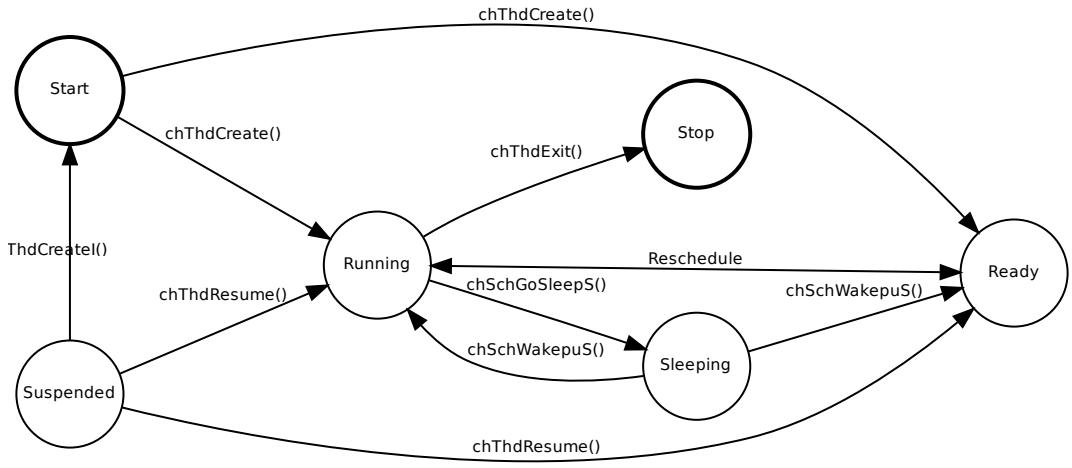


Currently Running Thread

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.

1.1.6 Thread States

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



1.1.7 Priority Levels

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

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

1.1.8 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 2

Topic Index

2.1 Topics

Here is a list of all topics with brief descriptions:

Compiler portability. . . . 21
Release and Licensing . . . 25
Release Information . . . 25
Customer Information . . . 27
License Settings . . . 30
RT Kernel . . . 33
Version Numbers and Identification . . . 34
Configuration . . . 36
Options . . . 36
Checks . . . 56
Restrictions . . . 56
System . . . 58
Port Interface . . . 58
OS Types and Structures . . . 59
OS Instances . . . 67
Runtime Faults Collection Unit . . . 70
Lists and Queues . . . 72
Scheduler . . . 84
Base Kernel Services . . . 103
System Management . . . 103
Memory Alignment . . . 126
Time and Intervals . . . 128
Virtual Timers . . . 141
Threads . . . 166
Time Measurement . . . 202
Synchronization . . . 206
Counting Semaphores . . . 206
Mutexes . . . 221
Condition Variables . . . 233
Event Flags . . . 243
Synchronous Messages . . . 266
Dynamic Threads . . . 277
Registry . . . 280
Debug . . . 289

Checks and Assertions . . . 289
Tracing . . . 299
Statistics . . . 310
OS Library . . . 314
Version Numbers and Identification . . . 314
Synchronization . . . 317
Binary Semaphores . . . 318
Mailboxes . . . 329
Pipes . . . 344
Delegate Threads . . . 354
Jobs Queues . . . 363
Memory Management . . . 375
Core Memory Manager . . . 375
Memory Heaps . . . 384
Memory Pools . . . 392
Complex Services . . . 413
Objects FIFOs . . . 413
Objects Caches . . . 427
Dynamic Objects Factory . . . 439

Chapter 3

Hierarchical Index

3.1 Class Hierarchy

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

ch_delta_list . . . 471
ch_dyn_element . . . 472
ch_dyn_list . . . 473
ch_dyn_mailbox . . . 474
ch_dyn_object . . . 475
ch_dyn_objects_fifo . . . 476
ch_dyn_pipe . . . 477
ch_dyn_semaphore . . . 479
ch_job_descriptor . . . 480
ch_jobs_queue . . . 481
ch_list . . . 483
ch_mutex . . . 484
ch_objects_cache . . . 485
ch_objects_factory . . . 489
ch_objects_fifo . . . 491
ch_oc_hash_header . . . 493
ch_oc_lru_header . . . 494
ch_oc_object . . . 496
ch_os_instance . . . 499
ch_os_instance_config . . . 503
ch_priority_queue . . . 504
ch_queue . . . 505
ch_ready_list . . . 506
ch_registered_static_object . . . 508
ch_registry . . . 509
ch_rfcu . . . 510
ch_semaphore . . . 511
    ch_binary_semaphore . . . 469
ch_system . . . 512
ch_system_debug . . . 514
ch_thread . . . 515
ch_threads_queue . . . 522
ch_virtual_timer . . . 523
ch_virtual_timers_list . . . 524
chdebug_t . . . 526

condition_variable 532
event_listener 533
event_source 535
guarded_memory_pool_t 536
heap_header 537
kernel_stats_t 539
mailbox_t 540
memcore_t 543
memory_heap 544
memory_pool_t 546
pipe_t 548
pool_header 551
thread_descriptor_t 551
time_measurement_t 554
tm_calibration_t 555
trace_buffer_t 556
trace_event_t 559

Chapter 4

Data Structure Index

4.1 Data Structures

Here are the data structures with brief descriptions:

ch_binary_semaphore
: Binary semaphore type . . . 469

ch_delta_list
: Delta list element and header structure . . . 471

ch_dyn_element
: Type of a dynamic object list element . . . 472

ch_dyn_list
: Type of a dynamic object list . . . 473

ch_dyn_mailbox
: Type of a dynamic buffer object . . . 474

ch_dyn_object
: Type of a dynamic buffer object . . . 475

ch_dyn_objects_fifo
: Type of a dynamic buffer object . . . 476

ch_dyn_pipe
: Type of a dynamic pipe object . . . 477

ch_dyn_semaphore
: Type of a dynamic semaphore . . . 479

ch_job_descriptor
: Type of a job descriptor . . . 480

ch_jobs_queue
: Type of a jobs queue . . . 481

ch_list
: Structure representing a generic single link list header and element . . . 483

ch_mutex
: Mutex structure . . . 484

ch_objects_cache
: Structure representing a cache object . . . 485

ch_objects_factory
: Type of the factory main object . . . 489

ch_objects_fifo
: Type of an objects FIFO . . . 491

ch_oc_hash_header
: Structure representing an hash table element . . . 493

ch_oc_lru_header
: Structure representing an hash table element . . . 494

ch_oc_object
Structure representing a cached object . . . 496
ch_os_instance
System instance data structure . . . 499
ch_os_instance_config
Type of an system instance configuration . . . 503
ch_priority_queue
Structure representing a generic priority-ordered bidirectional linked list header and element . . . 504
ch_queue
Structure representing a generic bidirectional linked list header and element . . . 505
ch_ready_list
Type of a ready list header . . . 506
ch_registered_static_object
Type of a registered object . . . 508
ch_registry
Type of a registry structure . . . 509
ch_rfcu
Type of an RFCU structure . . . 510
ch_semaphore
Semaphore structure . . . 511
ch_system
Type of system data structure . . . 512
ch_system_debug
System debug data structure . . . 514
ch_thread
Structure representing a thread . . . 515
ch_threads_queue
Type of a threads queue . . . 522
ch_virtual_timer
Structure representing a Virtual Timer . . . 523
ch_virtual_timers_list
Type of virtual timers list header . . . 524
chdebug_t
ChibiOS/RT memory signature record . . . 526
condition_variable
Condition_variable_t structure . . . 532
event_listener
Event Listener structure . . . 533
event_source
Event Source structure . . . 535
guarded_memory_pool_t
Guarded memory pool descriptor . . . 536
heap_header
Memory heap block header . . . 537
kernel_stats_t
Type of a kernel statistics structure . . . 539
mailbox_t
Structure representing a mailbox object . . . 540
memcore_t
Type of memory core object . . . 543
memory_heap
Structure describing a memory heap . . . 544
memory_pool_t
Memory pool descriptor . . . 546
pipe_t
Structure representing a pipe object . . . 548
pool_header
Memory pool free object header . . . 551

thread_descriptor_t
Type of a thread descriptor 551
time_measurement_t
Type of a Time Measurement object 554
tm_calibration_t
Type of a time measurement calibration data 555
trace_buffer_t
Trace buffer header 556
trace_event_t
Trace buffer record 559

Chapter 5

File Index

5.1 File List

Here is a list of all files with brief descriptions:

ccportab.h
Compiler portability layer . . . 563
chcustomer.h
Customer-related info . . . 564
chlicense.h
License Module macros and structures . . . 565
chversion.h
Version Module macros and structures . . . 566
chbsem.h
Binary semaphores structures and macros . . . 566
chdelegates.h
Delegate threads macros and structures . . . 568
chfactory.h
ChibiOS objects factory structures and macros . . . 569
chjobs.h
Jobs Queues structures and macros . . . 572
chlib.h
ChibiOS/LIB main include file . . . 574
chmboxes.h
Mailboxes macros and structures . . . 575
chmemcore.h
Core memory manager macros and structures . . . 576
chmemheaps.h
Memory heaps macros and structures . . . 577
chmempools.h
Memory Pools macros and structures . . . 578
chobjcaches.h
Objects Caches macros and structures . . . 580
chobjfifos.h
Objects FIFO structures and macros . . . 581
chpipes.h
Pipes macros and structures . . . 582
chdelegates.c
Delegate threads code . . . 583
chfactory.c
ChibiOS objects factory and registry code . . . 584

chmboxes.c
Mailboxes code . . . 586
chmemcore.c
Core memory manager code . . . 587
chmemheaps.c
Memory heaps code . . . 587
chmempools.c
Memory Pools code . . . 588
chobjcaches.c
Objects Caches code . . . 589
chpipes.c
Pipes code . . . 591
ch.h
ChibiOS/RT main include file . . . 592
chalign.h
Memory alignment macros and structures . . . 593
chchecks.h
Configuration file checks header . . . 594
chcond.h
Condition Variables macros and structures . . . 594
chdebug.h
Debug support macros and structures . . . 595
chdynamic.h
Dynamic threads macros and structures . . . 596
chearly.h
Early forward types declarations header . . . 596
chevents.h
Events macros and structures . . . 597
chinstances.h
OS instances macros and structures . . . 599
chlists.h
Lists and Queues header . . . 600
chmsg.h
Messages macros and structures . . . 602
chmtx.h
Mutexes macros and structures . . . 602
chobjects.h
Operating System Objects macros and structures . . . 604
chport.h
Port wrapper header . . . 605
chregistry.h
Threads registry macros and structures . . . 605
chrestrictions.h
Licensing restrictions header . . . 606
chrfcu.h
Runtime Faults Collection Unit macros and structures . . . 607
chschd.h
Scheduler macros and structures . . . 607
chsem.h
Semaphores macros and structures . . . 610
chstats.h
Statistics module macros and structures . . . 611
chsys.h
System related macros and structures . . . 611
chthreads.h
Threads module macros and structures . . . 613
chtime.h
Time and intervals macros and structures . . . 616

chtm.h
Time Measurement module macros and structures . . . 618
chtrace.h
Tracer macros and structures . . . 618
chvt.h
Time and Virtual Timers module macros and structures . . . 620
chcond.c
Condition Variables code . . . 622
chdebug.c
Debug support code . . . 622
chdynamic.c
Dynamic threads code . . . 623
chevents.c
Events code . . . 624
chinstances.c
OS instances code . . . 625
chmsg.c
Messages code . . . 625
chmtx.c
Mutexes code . . . 626
chregistry.c
Threads registry code . . . 626
chrfcu.c
Runtime Faults Collection Unit code . . . 627
chschd.c
Scheduler code . . . 627
chsem.c
Semaphores code . . . 628
chstats.c
Statistics module code . . . 629
chsys.c
System related code . . . 630
chthreads.c
Threads code . . . 631
chtm.c
Time Measurement module code . . . 632
chtrace.c
Tracer code . . . 633
chvt.c
Time and Virtual Timers module code . . . 634
chconf.h
Configuration file template . . . 634

Chapter 6

Topic Documentation

6.1 Compiler portability.

6.1.1 Detailed Description

Compiler abstraction macros

- #define CC_SECTION(s)

  Allocates a variable or function to a specific section.
- #define CC_WEAK __attribute__((weak))

  Marks a function or variable as a weak symbol.
- #define CC_USED __attribute__((used))

  Marks a function or variable as used.
- #define CC_ALIGN_DATA(n)

  Enforces alignment of the variable declared afterward.
- #define CC_ALIGN_CODE(n)

  Enforces alignment of a function declared afterward.
- #define CC_PACK __attribute__((packed))

  Enforces packing of the structure declared afterward.
- #define CC_NO_INLINE __attribute__((noinline))

  Marks a function as not inlineable.
- #define CC_FORCE_INLINE __attribute__((always_inline))

  Enforces a function inline.
- #define CC_NO_RETURN __attribute__((noreturn))

  Marks a function as non-returning.
- #define CC_ROMCONST const

  Enforces a variable in a ROM area.
- #define CC_LIKELY(x)

  Marks a boolean expression as likely true.
- #define CC_UNLIKELY(x)

  Marks a boolean expression as likely false.

6.1.2 Macro Definition Documentation

6.1.2.1 CC_SECTION

```
#define CC_SECTION(
              s)
```

Value:

```
__attribute__((section(s)))
```

Allocates a variable or function to a specific section.

Note

If the compiler does not support such a feature then this macro must not be defined or it could originate errors.

6.1.2.2 CC_WEAK

```
#define CC_WEAK __attribute__((weak))
```

Marks a function or variable as a weak symbol.

Note

If the compiler does not support such a feature then this macro must not be defined or it could originate errors.

6.1.2.3 CC_USED

```
#define CC_USED __attribute__((used))
```

Marks a function or variable as used.

The compiler or linker shall not remove the marked function or variable regardless if it is referred or not in the code.

Note

If the compiler does not support such a feature then this macro must not be defined or it could originate errors.

6.1.2.4 CC_ALIGN_DATA

```
#define CC_ALIGN_DATA(
              n)
```

Value:

```
__attribute__((aligned(n)))
```

Enforces alignment of the variable declared afterward.

Note

If the compiler does not support such a feature then this macro must not be defined or it could originate errors.

6.1.2.5 CC_ALIGN_CODE

```
#define CC_ALIGN_CODE(
              n)
```

Value:

```
__attribute__((aligned(n)))
```

Enforces alignment of a function declared afterward.

Note

If the compiler does not support such a feature then this macro must not be defined or it could originate errors.

6.1.2.6 CC_PACK

```
#define CC_PACK __attribute__((packed))
```

Enforces packing of the structure declared afterward.

Note

If the compiler does not support such a feature then this macro must not be defined or it could originate errors.

6.1.2.7 CC_NO_INLINE

```
#define CC_NO_INLINE __attribute__((noinline))
```

Marks a function as not inlineable.

Note

Can be implemented as an empty macro if not supported by the compiler.

6.1.2.8 CC_FORCE_INLINE

```
#define CC_FORCE_INLINE __attribute__((always_inline))
```

Enforces a function inline.

Note

Can be implemented as an empty macro if not supported by the compiler.

6.1.2.9 CC_NO_RETURN

```
#define CC_NO_RETURN __attribute__((noreturn))
```

Marks a function as non-returning.

Note

Can be implemented as an empty macro if not supported by the compiler.

6.1.2.10 CC_ROMCONST

```
#define CC_ROMCONST const
```

Enforces a variable in a ROM area.

Note

Can be implemented as an empty macro if not supported by the compiler.

6.1.2.11 CC_LIKELY

```
#define CC_LIKELY(
              x)
```

Value:

```
__builtin_expect(!!(x), 1)
```

Marks a boolean expression as likely true.

Parameters

in	*x*	a valid expression

6.1.2.12 CC_UNLIKELY

```
#define CC_UNLIKELY(
              x)
```

Value:

```
__builtin_expect(!!(x), 0)
```

Marks a boolean expression as likely false.

Parameters

in	*x*	a valid expression

6.2 Release and Licensing

6.2.1 Detailed Description

Topics

- Release Information
- Customer Information
- License Settings

6.2.2 Release Information

6.2.2.1 Detailed Description

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

ChibiOS version identification

- #define CH_VERSION "2021.11.4"

  ChibiOS version string.
- #define CH_VERSION_YEAR 21

  ChibiOS version release year.
- #define CH_VERSION_MONTH 11

  ChibiOS version release month.
- #define CH_VERSION_PATCH 4

  ChibiOS version patch number.
- #define CH_VERSION_NICKNAME "Paestum"

  ChibiOS version nickname.

Macros

- #define __CHIBIOS__

  ChibiOS product identification macro.
- #define CH_VERSION_STABLE 1

  Stable release flag.
- #define CH_VERSION_DATE (((CH_VERSION_YEAR + 2000) * 100) + CH_VERSION_MONTH)

  Current version date in numeric form (yyyymm).

6.2.2.2 Macro Definition Documentation

6.2.2.2.1 __CHIBIOS__

```
#define __CHIBIOS__
```

ChibiOS product identification macro.

6.2.2.2.2 CH_VERSION_STABLE

```
#define CH_VERSION_STABLE 1
```

Stable release flag.

6.2.2.2.3 CH_VERSION

```
#define CH_VERSION "2021.11.4"
```

ChibiOS version string.

6.2.2.2.4 CH_VERSION_YEAR

```
#define CH_VERSION_YEAR 21
```

ChibiOS version release year.

6.2.2.2.5 CH_VERSION_MONTH

```
#define CH_VERSION_MONTH 11
```

ChibiOS version release month.

6.2.2.2.6 CH_VERSION_PATCH

```
#define CH_VERSION_PATCH 4
```

ChibiOS version patch number.

6.2.2.2.7 CH_VERSION_NICKNAME

```
#define CH_VERSION_NICKNAME "Paestum"
```

ChibiOS version nickname.

6.2.2.2.8 CH_VERSION_DATE

```
#define CH_VERSION_DATE  (((CH_VERSION_YEAR + 2000) * 100) + CH_VERSION_MONTH)
```

Current version date in numeric form (yyyymm).

6.2.3 Customer Information

6.2.3.1 Detailed Description

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

Licensed Products

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

Macros

- #define CH_CUSTOMER_ID_STRING "Santa, North Pole"

  Customer readable identifier.
- #define CH_CUSTOMER_ID_CODE "xxxx-yyyy"

  Customer code.
- #define CH_CUSTOMER_LICENSE_EOS_DATE 209912

  End-Of-Support date (yyyymm).
- #define CH_CUSTOMER_LICENSE_VERSION_YEAR 99

  Licensed branch year.
- #define CH_CUSTOMER_LICENSE_VERSION_MONTH 12

  Licensed branch month.
- #define CH_LICENSE CH_LICENSE_GPL

  Current license.
- #define CH_CUSTOMER_LICENSE_VERSION_DATE

  Licensed version date in numeric form (yyyymm).

6.2.3.2 Macro Definition Documentation

6.2.3.2.1 CH_CUSTOMER_ID_STRING

```
#define CH_CUSTOMER_ID_STRING "Santa, North Pole"
```

Customer readable identifier.

6.2.3.2.2 CH_CUSTOMER_ID_CODE

```
#define CH_CUSTOMER_ID_CODE "xxxx-yyyy"
```

Customer code.

6.2.3.2.3 CH_CUSTOMER_LICENSE_EOS_DATE

```
#define CH_CUSTOMER_LICENSE_EOS_DATE 209912
```

End-Of-Support date (yyyymm).

6.2.3.2.4 CH_CUSTOMER_LICENSE_VERSION_YEAR

```
#define CH_CUSTOMER_LICENSE_VERSION_YEAR 99
```

Licensed branch year.

6.2.3.2.5 CH_CUSTOMER_LICENSE_VERSION_MONTH

```
#define CH_CUSTOMER_LICENSE_VERSION_MONTH 12
```

Licensed branch month.

6.2.3.2.6 CH_LICENSE

```
#define CH_LICENSE CH_LICENSE_GPL
```

Current license.

Note

This setting is reserved to the copyright owner.

Changing this setting invalidates the license.

The license statement in the source headers is valid, applicable and binding regardless this setting.

6.2.3.2.7 CH_CUSTOMER_LIC_RT

```
#define CH_CUSTOMER_LIC_RT TRUE
```

6.2.3.2.8 CH_CUSTOMER_LIC_NIL

```
#define CH_CUSTOMER_LIC_NIL TRUE
```

6.2.3.2.9 CH_CUSTOMER_LIC_OSLIB

```
#define CH_CUSTOMER_LIC_OSLIB TRUE
```

6.2.3.2.10 CH_CUSTOMER_LIC_EX

```
#define CH_CUSTOMER_LIC_EX TRUE
```

6.2.3.2.11 CH_CUSTOMER_LIC_SB

```
#define CH_CUSTOMER_LIC_SB TRUE
```

6.2.3.2.12 CH_CUSTOMER_LIC_PORT_CM0

```
#define CH_CUSTOMER_LIC_PORT_CM0 TRUE
```

6.2.3.2.13 CH_CUSTOMER_LIC_PORT_CM3

```
#define CH_CUSTOMER_LIC_PORT_CM3 TRUE
```

6.2.3.2.14 CH_CUSTOMER_LIC_PORT_CM4

```
#define CH_CUSTOMER_LIC_PORT_CM4 TRUE
```

6.2.3.2.15 CH_CUSTOMER_LIC_PORT_CM7

```
#define CH_CUSTOMER_LIC_PORT_CM7 TRUE
```

6.2.3.2.16 CH_CUSTOMER_LIC_PORT_CM33

```
#define CH_CUSTOMER_LIC_PORT_CM33 TRUE
```

6.2.3.2.17 CH_CUSTOMER_LIC_PORT_ARM79

```
#define CH_CUSTOMER_LIC_PORT_ARM79 TRUE
```

6.2.3.2.18 CH_CUSTOMER_LIC_PORT_E200Z0

```
#define CH_CUSTOMER_LIC_PORT_E200Z0 TRUE
```

6.2.3.2.19 CH_CUSTOMER_LIC_PORT_E200Z2

```
#define CH_CUSTOMER_LIC_PORT_E200Z2 TRUE
```

6.2.3.2.20 CH_CUSTOMER_LIC_PORT_E200Z3

```
#define CH_CUSTOMER_LIC_PORT_E200Z3 TRUE
```

6.2.3.2.21 CH_CUSTOMER_LIC_PORT_E200Z4

```
#define CH_CUSTOMER_LIC_PORT_E200Z4 TRUE
```

6.2.3.2.22 CH_CUSTOMER_LICENSE_VERSION_DATE

```
#define CH_CUSTOMER_LICENSE_VERSION_DATE
```

Value:

```
(((CH_CUSTOMER_LICENSE_VERSION_YEAR + 2000) * 100) +                    \
 CH_CUSTOMER_LICENSE_VERSION_MONTH)
```

Licensed version date in numeric form (yyyymm).

6.2.4 License Settings

6.2.4.1 Detailed Description

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

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

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.

6.2.4.2 Macro Definition Documentation

6.2.4.2.1 CH_FEATURES_BASIC

```
#define CH_FEATURES_BASIC 0
```

6.2.4.2.2 CH_FEATURES_INTERMEDIATE

```
#define CH_FEATURES_INTERMEDIATE 1
```

6.2.4.2.3 CH_FEATURES_FULL

```
#define CH_FEATURES_FULL 2
```

6.2.4.2.4 CH_DEPLOY_UNLIMITED

```
#define CH_DEPLOY_UNLIMITED -1
```

6.2.4.2.5 CH_DEPLOY_NONE

```
#define CH_DEPLOY_NONE 0
```

6.2.4.2.6 CH_LICENSE_GPL

```
#define CH_LICENSE_GPL 0
```

6.2.4.2.7 CH_LICENSE_GPL_EXCEPTION

```
#define CH_LICENSE_GPL_EXCEPTION 1
```

6.2.4.2.8 CH_LICENSE_COMMERCIAL_FREE

```
#define CH_LICENSE_COMMERCIAL_FREE 2
```

6.2.4.2.9 CH_LICENSE_COMMERCIAL_DEV_1000

```
#define CH_LICENSE_COMMERCIAL_DEV_1000 3
```

6.2.4.2.10 CH_LICENSE_COMMERCIAL_DEV_5000

```
#define CH_LICENSE_COMMERCIAL_DEV_5000 4
```

6.2.4.2.11 CH_LICENSE_COMMERCIAL_FULL

```
#define CH_LICENSE_COMMERCIAL_FULL 5
```

6.2.4.2.12 CH_LICENSE_COMMERCIAL_RUNTIME

```
#define CH_LICENSE_COMMERCIAL_RUNTIME 6
```

6.2.4.2.13 CH_LICENSE_PARTNER

```
#define CH_LICENSE_PARTNER 7
```

6.2.4.2.14 CH_LICENSE_TYPE_STRING

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

License identification string.

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

6.2.4.2.15 CH_LICENSE_ID_STRING

```
#define CH_LICENSE_ID_STRING "N/A"
```

Customer identification string.

This information is only available for registered commercial users.

6.2.4.2.16 CH_LICENSE_ID_CODE

```
#define CH_LICENSE_ID_CODE "N/A"
```

Customer code.

This information is only available for registered commercial users.

6.2.4.2.17 CH_LICENSE_MODIFIABLE_CODE

```
#define CH_LICENSE_MODIFIABLE_CODE TRUE
```

Code modifiability restrictions.

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

6.2.4.2.18 CH_LICENSE_FEATURES

```
#define CH_LICENSE_FEATURES CH_FEATURES_FULL
```

Code functionality restrictions.

6.2.4.2.19 CH_LICENSE_MAX_DEPLOY

```
#define CH_LICENSE_MAX_DEPLOY CH_DEPLOY_UNLIMITED
```

Code deploy restrictions.

This is the per-core deploy limit allowed under the current license scheme.

6.3 RT Kernel

6.3.1 Detailed Description

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

Topics

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

6.3.2 Version Numbers and Identification

6.3.2.1 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.

ChibiOS/RT version identification

- #define CH_KERNEL_VERSION "7.0.5"
  Kernel version string.
- #define CH_KERNEL_MAJOR 7
  Kernel version major number.
- #define CH_KERNEL_MINOR 0
  Kernel version minor number.
- #define CH_KERNEL_PATCH 5
  Kernel version patch number.

Constants for configuration options

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

Macros

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

6.3.2.2 Macro Definition Documentation

6.3.2.2.1 __CHIBIOS_RT__

```
#define __CHIBIOS_RT__
```

ChibiOS/RT identification macro.

6.3.2.2.2 CH_KERNEL_STABLE

```
#define CH_KERNEL_STABLE 1
```

Stable release flag.

6.3.2.2.3 CH_KERNEL_VERSION

```
#define CH_KERNEL_VERSION "7.0.5"
```

Kernel version string.

6.3.2.2.4 CH_KERNEL_MAJOR

```
#define CH_KERNEL_MAJOR 7
```

Kernel version major number.

6.3.2.2.5 CH_KERNEL_MINOR

```
#define CH_KERNEL_MINOR 0
```

Kernel version minor number.

6.3.2.2.6 CH_KERNEL_PATCH

```
#define CH_KERNEL_PATCH 5
```

Kernel version patch number.

6.3.2.2.7 FALSE

```
#define FALSE 0
```

Generic 'false' preprocessor boolean constant.

Note

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

6.3.2.2.8 TRUE

```
#define TRUE 1
```

Generic 'true' preprocessor boolean constant.

Note

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

6.3.3 Configuration

6.3.3.1 Detailed Description

Topics

- Options
- Checks
- Restrictions

6.3.3.2 Options

6.3.3.2.1 Detailed Description

Kernel related settings and hooks.

System settings

- #define CH_CFG_SMP_MODE FALSE

  Handling of instances.

System timers settings

- #define CH_CFG_ST_RESOLUTION 32

  System time counter resolution.
- #define CH_CFG_ST_FREQUENCY 10000

  System tick frequency.
- #define CH_CFG_INTERVALS_SIZE 32

  Time intervals data size.
- #define CH_CFG_TIME_TYPES_SIZE 32

  Time types data size.
- #define CH_CFG_ST_TIMEDELTA 2

  Time delta constant for the tick-less mode.

Kernel parameters and options

- #define CH_CFG_TIME_QUANTUM 0

  Round robin interval.
- #define CH_CFG_NO_IDLE_THREAD FALSE

  Idle thread automatic spawn suppression.

Performance options

- #define CH_CFG_OPTIMIZE_SPEED TRUE

  OS optimization.

Subsystem options

- #define CH_CFG_USE_TM TRUE

  Time Measurement APIs.
- #define CH_CFG_USE_TIMESTAMP TRUE

  Time Stamps APIs.
- #define CH_CFG_USE_REGISTRY TRUE

  Threads registry APIs.
- #define CH_CFG_USE_WAITEXIT TRUE

  Threads synchronization APIs.
- #define CH_CFG_USE_SEMAPHORES TRUE

  Semaphores APIs.
- #define CH_CFG_USE_SEMAPHORES_PRIORITY FALSE

  Semaphores queuing mode.
- #define CH_CFG_USE_MUTEXES TRUE

  Mutexes APIs.
- #define CH_CFG_USE_MUTEXES_RECURSIVE FALSE

  Enables recursive behavior on mutexes.
- #define CH_CFG_USE_CONDVARS TRUE

  Conditional Variables APIs.
- #define CH_CFG_USE_CONDVARS_TIMEOUT TRUE

  Conditional Variables APIs with timeout.
- #define CH_CFG_USE_EVENTS TRUE

  Events Flags APIs.
- #define CH_CFG_USE_EVENTS_TIMEOUT TRUE

  Events Flags APIs with timeout.
- #define CH_CFG_USE_MESSAGES TRUE

  Synchronous Messages APIs.
- #define CH_CFG_USE_MESSAGES_PRIORITY FALSE

  Synchronous Messages queuing mode.
- #define CH_CFG_USE_DYNAMIC TRUE

  Dynamic Threads APIs.

OSLIB options

- #define CH_CFG_USE_MAILBOXES TRUE

  Mailboxes APIs.
- #define CH_CFG_USE_MEMCORE TRUE

  Core Memory Manager APIs.
- #define CH_CFG_MEMCORE_SIZE 0

  Managed RAM size.
- #define CH_CFG_USE_HEAP TRUE

  Heap Allocator APIs.
- #define CH_CFG_USE_MEMPOOLS TRUE

  Memory Pools Allocator APIs.
- #define CH_CFG_USE_OBJ_FIFOS TRUE

  Objects FIFOs APIs.
- #define CH_CFG_USE_PIPES TRUE

  Pipes APIs.
- #define CH_CFG_USE_OBJ_CACHES TRUE

  Objects Caches APIs.
- #define CH_CFG_USE_DELEGATES TRUE

  Delegate threads APIs.
- #define CH_CFG_USE_JOBS TRUE

  Jobs Queues APIs.

Objects factory options

- #define CH_CFG_USE_FACTORY TRUE

  Objects Factory APIs.
- #define CH_CFG_FACTORY_MAX_NAMES_LENGTH 8

  Maximum length for object names.
- #define CH_CFG_FACTORY_OBJECTS_REGISTRY TRUE

  Enables the registry of generic objects.
- #define CH_CFG_FACTORY_GENERIC_BUFFERS TRUE

  Enables factory for generic buffers.
- #define CH_CFG_FACTORY_SEMAPHORES TRUE

  Enables factory for semaphores.
- #define CH_CFG_FACTORY_MAILBOXES TRUE

  Enables factory for mailboxes.
- #define CH_CFG_FACTORY_OBJ_FIFOS TRUE

  Enables factory for objects FIFOs.
- #define CH_CFG_FACTORY_PIPES TRUE

  Enables factory for Pipes.

Debug options

- #define CH_DBG_STATISTICS FALSE

  Debug option, kernel statistics.
- #define CH_DBG_SYSTEM_STATE_CHECK TRUE

  Debug option, system state check.
- #define CH_DBG_ENABLE_CHECKS TRUE

  Debug option, parameters checks.
- #define CH_DBG_ENABLE_ASSERTS TRUE

  Debug option, consistency checks.
- #define CH_DBG_TRACE_MASK CH_DBG_TRACE_MASK_ALL

  Debug option, trace buffer.
- #define CH_DBG_TRACE_BUFFER_SIZE 128

  Trace buffer entries.
- #define CH_DBG_ENABLE_STACK_CHECK TRUE

  Debug option, stack checks.
- #define CH_DBG_FILL_THREADS TRUE

  Debug option, stacks initialization.
- #define CH_DBG_THREADS_PROFILING FALSE

  Debug option, threads profiling.

Kernel hooks

- #define CH_CFG_SYSTEM_EXTRA_FIELDS /∗ Add system custom fields here.∗/

  System structure extension.
- #define CH_CFG_SYSTEM_INIT_HOOK()

  System initialization hook.
- #define CH_CFG_OS_INSTANCE_EXTRA_FIELDS /∗ Add OS instance custom fields here.∗/

  OS instance structure extension.
- #define CH_CFG_OS_INSTANCE_INIT_HOOK(oip)

  OS instance initialization hook.

- #define CH_CFG_THREAD_EXTRA_FIELDS /* Add threads custom fields here.*/

  Threads descriptor structure extension.
- #define CH_CFG_THREAD_INIT_HOOK(tp)

  Threads initialization hook.
- #define CH_CFG_THREAD_EXIT_HOOK(tp)

  Threads finalization hook.
- #define CH_CFG_CONTEXT_SWITCH_HOOK(ntp, otp)

  Context switch hook.
- #define CH_CFG_IRQ_PROLOGUE_HOOK()

  ISR enter hook.
- #define CH_CFG_IRQ_EPILOGUE_HOOK()

  ISR exit hook.
- #define CH_CFG_IDLE_ENTER_HOOK()

  Idle thread enter hook.
- #define CH_CFG_IDLE_LEAVE_HOOK()

  Idle thread leave hook.
- #define CH_CFG_IDLE_LOOP_HOOK()

  Idle Loop hook.
- #define CH_CFG_SYSTEM_TICK_HOOK()

  System tick event hook.
- #define CH_CFG_SYSTEM_HALT_HOOK(reason)

  System halt hook.
- #define CH_CFG_TRACE_HOOK(tep)

  Trace hook.
- #define CH_CFG_RUNTIME_FAULTS_HOOK(mask)

  Runtime Faults Collection Unit hook.

Macros

- #define _CHIBIOS_RT_CONF_
- #define _CHIBIOS_RT_CONF_VER_7_0_

6.3.3.2.2 Macro Definition Documentation

6.3.3.2.2.1 _CHIBIOS_RT_CONF_

```
#define _CHIBIOS_RT_CONF_
```

6.3.3.2.2.2 _CHIBIOS_RT_CONF_VER_7_0_

```
#define _CHIBIOS_RT_CONF_VER_7_0_
```

6.3.3.2.2.3 CH_CFG_SMP_MODE

```
#define CH_CFG_SMP_MODE FALSE
```

Handling of instances.

Note

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

6.3.3.2.2.4 CH_CFG_ST_RESOLUTION

```
#define CH_CFG_ST_RESOLUTION 32
```

System time counter resolution.

Note

Allowed values are 16, 32 or 64 bits.

6.3.3.2.2.5 CH_CFG_ST_FREQUENCY

```
#define CH_CFG_ST_FREQUENCY 10000
```

System tick frequency.

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

6.3.3.2.2.6 CH_CFG_INTERVALS_SIZE

```
#define CH_CFG_INTERVALS_SIZE 32
```

Time intervals data size.

Note

Allowed values are 16, 32 or 64 bits.

6.3.3.2.2.7 CH_CFG_TIME_TYPES_SIZE

```
#define CH_CFG_TIME_TYPES_SIZE 32
```

Time types data size.

Note

Allowed values are 16 or 32 bits.

6.3.3.2.2.8 CH_CFG_ST_TIMEDELTA

```
#define CH_CFG_ST_TIMEDELTA 2
```

Time delta constant for the tick-less mode.

Note

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

6.3.3.2.2.9 CH_CFG_TIME_QUANTUM

```
#define CH_CFG_TIME_QUANTUM 0
```

Round robin interval.

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

Note

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

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

6.3.3.2.2.10 CH_CFG_NO_IDLE_THREAD

```
#define CH_CFG_NO_IDLE_THREAD FALSE
```

Idle thread automatic spawn suppression.

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

6.3.3.2.2.11 CH_CFG_OPTIMIZE_SPEED

```
#define CH_CFG_OPTIMIZE_SPEED TRUE
```

OS optimization.

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

Note

This is not related to the compiler optimization options.

The default is `TRUE`.

6.3.3.2.2.12 CH_CFG_USE_TM

```
#define CH_CFG_USE_TM TRUE
```

Time Measurement APIs.

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

Note

The default is `TRUE`.

6.3.3.2.2.13 CH_CFG_USE_TIMESTAMP

```
#define CH_CFG_USE_TIMESTAMP TRUE
```

Time Stamps APIs.

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

Note

The default is `TRUE`.

6.3.3.2.2.14 CH_CFG_USE_REGISTRY

```
#define CH_CFG_USE_REGISTRY TRUE
```

Threads registry APIs.

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

Note

The default is `TRUE`.

6.3.3.2.2.15 CH_CFG_USE_WAITEXIT

```
#define CH_CFG_USE_WAITEXIT TRUE
```

Threads synchronization APIs.

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

Note

The default is `TRUE`.

6.3.3.2.2.16 CH_CFG_USE_SEMAPHORES

```
#define CH_CFG_USE_SEMAPHORES TRUE
```

Semaphores APIs.

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

Note

The default is `TRUE`.

6.3.3.2.2.17 CH_CFG_USE_SEMAPHORES_PRIORITY

```
#define CH_CFG_USE_SEMAPHORES_PRIORITY FALSE
```

Semaphores queuing mode.

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

Note

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

Requires `CH_CFG_USE_SEMAPHORES`.

6.3.3.2.2.18 CH_CFG_USE_MUTEXES

```
#define CH_CFG_USE_MUTEXES TRUE
```

Mutexes APIs.

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

Note

The default is `TRUE`.

6.3.3.2.2.19 CH_CFG_USE_MUTEXES_RECURSIVE

```
#define CH_CFG_USE_MUTEXES_RECURSIVE FALSE
```

Enables recursive behavior on mutexes.

Note

Recursive mutexes are heavier and have an increased memory footprint.

The default is `FALSE`.

Requires `CH_CFG_USE_MUTEXES`.

6.3.3.2.2.20 CH_CFG_USE_CONDVARS

```
#define CH_CFG_USE_CONDVARS TRUE
```

Conditional Variables APIs.

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

Note

The default is `TRUE`.
Requires `CH_CFG_USE_MUTEXES`.

6.3.3.2.2.21 CH_CFG_USE_CONDVARS_TIMEOUT

```
#define CH_CFG_USE_CONDVARS_TIMEOUT TRUE
```

Conditional Variables APIs with timeout.

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

Note

The default is `TRUE`.
Requires `CH_CFG_USE_CONDVARS`.

6.3.3.2.2.22 CH_CFG_USE_EVENTS

```
#define CH_CFG_USE_EVENTS TRUE
```

Events Flags APIs.

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

Note

The default is `TRUE`.

6.3.3.2.2.23 CH_CFG_USE_EVENTS_TIMEOUT

```
#define CH_CFG_USE_EVENTS_TIMEOUT TRUE
```

Events Flags APIs with timeout.

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

Note

The default is `TRUE`.
Requires `CH_CFG_USE_EVENTS`.

6.3.3.2.2.24 CH_CFG_USE_MESSAGES

```
#define CH_CFG_USE_MESSAGES TRUE
```

Synchronous Messages APIs.

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

Note

The default is `TRUE`.

6.3.3.2.2.25 CH_CFG_USE_MESSAGES_PRIORITY

```
#define CH_CFG_USE_MESSAGES_PRIORITY FALSE
```

Synchronous Messages queuing mode.

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

Note

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

6.3.3.2.2.26 CH_CFG_USE_DYNAMIC

```
#define CH_CFG_USE_DYNAMIC TRUE
```

Dynamic Threads APIs.

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

Note

The default is `TRUE`.
Requires `CH_CFG_USE_WAITEXIT`.
Requires `CH_CFG_USE_HEAP` and/or `CH_CFG_USE_MEMPOOLS`.

6.3.3.2.2.27 CH_CFG_USE_MAILBOXES

```
#define CH_CFG_USE_MAILBOXES TRUE
```

Mailboxes APIs.

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

Note

The default is `TRUE`.
Requires `CH_CFG_USE_SEMAPHORES`.

6.3.3.2.2.28 CH_CFG_USE_MEMCORE

```
#define CH_CFG_USE_MEMCORE TRUE
```

Core Memory Manager APIs.

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

Note

The default is TRUE.

6.3.3.2.2.29 CH_CFG_MEMCORE_SIZE

```
#define CH_CFG_MEMCORE_SIZE 0
```

Managed RAM size.

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

Note

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

Requires CH_CFG_USE_MEMCORE.

6.3.3.2.2.30 CH_CFG_USE_HEAP

```
#define CH_CFG_USE_HEAP TRUE
```

Heap Allocator APIs.

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

Note

The default is TRUE.

Requires CH_CFG_USE_MEMCORE and either CH_CFG_USE_MUTEXES or CH_CFG_USE_↩SEMAPHORES.

Mutexes are recommended.

6.3.3.2.2.31 CH_CFG_USE_MEMPOOLS

```
#define CH_CFG_USE_MEMPOOLS TRUE
```

Memory Pools Allocator APIs.

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

Note

The default is TRUE.

6.3.3.2.2.32 CH_CFG_USE_OBJ_FIFOS

```
#define CH_CFG_USE_OBJ_FIFOS TRUE
```

Objects FIFOs APIs.

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

Note

The default is `TRUE`.

6.3.3.2.2.33 CH_CFG_USE_PIPES

```
#define CH_CFG_USE_PIPES TRUE
```

Pipes APIs.

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

Note

The default is `TRUE`.

6.3.3.2.2.34 CH_CFG_USE_OBJ_CACHES

```
#define CH_CFG_USE_OBJ_CACHES TRUE
```

Objects Caches APIs.

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

Note

The default is `TRUE`.

6.3.3.2.2.35 CH_CFG_USE_DELEGATES

```
#define CH_CFG_USE_DELEGATES TRUE
```

Delegate threads APIs.

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

Note

The default is `TRUE`.

6.3.3.2.2.36 CH_CFG_USE_JOBS

```
#define CH_CFG_USE_JOBS TRUE
```

Jobs Queues APIs.

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

Note

The default is `TRUE`.

6.3.3.2.2.37 CH_CFG_USE_FACTORY

```
#define CH_CFG_USE_FACTORY TRUE
```

Objects Factory APIs.

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

Note

The default is `FALSE`.

6.3.3.2.2.38 CH_CFG_FACTORY_MAX_NAMES_LENGTH

```
#define CH_CFG_FACTORY_MAX_NAMES_LENGTH 8
```

Maximum length for object names.

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

6.3.3.2.2.39 CH_CFG_FACTORY_OBJECTS_REGISTRY

```
#define CH_CFG_FACTORY_OBJECTS_REGISTRY TRUE
```

Enables the registry of generic objects.

6.3.3.2.2.40 CH_CFG_FACTORY_GENERIC_BUFFERS

```
#define CH_CFG_FACTORY_GENERIC_BUFFERS TRUE
```

Enables factory for generic buffers.

6.3.3.2.2.41 CH_CFG_FACTORY_SEMAPHORES

```
#define CH_CFG_FACTORY_SEMAPHORES TRUE
```

Enables factory for semaphores.

6.3.3.2.2.42 CH_CFG_FACTORY_MAILBOXES

```
#define CH_CFG_FACTORY_MAILBOXES TRUE
```

Enables factory for mailboxes.

6.3.3.2.2.43 CH_CFG_FACTORY_OBJ_FIFOS

```
#define CH_CFG_FACTORY_OBJ_FIFOS TRUE
```

Enables factory for objects FIFOs.

6.3.3.2.2.44 CH_CFG_FACTORY_PIPES

```
#define CH_CFG_FACTORY_PIPES TRUE
```

Enables factory for Pipes.

6.3.3.2.2.45 CH_DBG_STATISTICS

```
#define CH_DBG_STATISTICS FALSE
```

Debug option, kernel statistics.

Note

The default is `FALSE`.

6.3.3.2.2.46 CH_DBG_SYSTEM_STATE_CHECK

```
#define CH_DBG_SYSTEM_STATE_CHECK TRUE
```

Debug option, system state check.

If enabled the correct call protocol for system APIs is checked at runtime.

Note

The default is `FALSE`.

6.3.3.2.2.47 CH_DBG_ENABLE_CHECKS

```
#define CH_DBG_ENABLE_CHECKS TRUE
```

Debug option, parameters checks.

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

Note

The default is `FALSE`.

6.3.3.2.2.48 CH_DBG_ENABLE_ASSERTS

```
#define CH_DBG_ENABLE_ASSERTS TRUE
```

Debug option, consistency checks.

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

Note

The default is FALSE.

6.3.3.2.2.49 CH_DBG_TRACE_MASK

```
#define CH_DBG_TRACE_MASK CH_DBG_TRACE_MASK_ALL
```

Debug option, trace buffer.

If enabled then the trace buffer is activated.

Note

The default is CH_DBG_TRACE_MASK_DISABLED.

6.3.3.2.2.50 CH_DBG_TRACE_BUFFER_SIZE

```
#define CH_DBG_TRACE_BUFFER_SIZE 128
```

Trace buffer entries.

Note

The trace buffer is only allocated if CH_DBG_TRACE_MASK is different from CH_DBG_TRACE_MASK_↩ DISABLED.

6.3.3.2.2.51 CH_DBG_ENABLE_STACK_CHECK

```
#define CH_DBG_ENABLE_STACK_CHECK TRUE
```

Debug option, stack checks.

If enabled then a runtime stack check is performed.

Note

The default is FALSE.

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

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

6.3.3.2.2.52 CH_DBG_FILL_THREADS

```
#define CH_DBG_FILL_THREADS TRUE
```

Debug option, stacks initialization.

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

Note

The default is FALSE.

6.3.3.2.2.53 CH_DBG_THREADS_PROFILING

```
#define CH_DBG_THREADS_PROFILING FALSE
```

Debug option, threads profiling.

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

Note

The default is FALSE.

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

6.3.3.2.2.54 CH_CFG_SYSTEM_EXTRA_FIELDS

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

System structure extension.

User fields added to the end of the ch_system_t structure.

6.3.3.2.2.55 CH_CFG_SYSTEM_INIT_HOOK

```
#define CH_CFG_SYSTEM_INIT_HOOK()
```

Value:

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

System initialization hook.

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

6.3.3.2.2.56 CH_CFG_OS_INSTANCE_EXTRA_FIELDS

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

OS instance structure extension.

User fields added to the end of the os_instance_t structure.

6.3.3.2.2.57 CH_CFG_OS_INSTANCE_INIT_HOOK

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

Value:

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

OS instance initialization hook.

Parameters

in	*oip*	pointer to the os_instance_t structure

6.3.3.2.2.58 CH_CFG_THREAD_EXTRA_FIELDS

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

Threads descriptor structure extension.

User fields added to the end of the thread_t structure.

6.3.3.2.2.59 CH_CFG_THREAD_INIT_HOOK

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

Value:

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

Threads initialization hook.

User initialization code added to the _thread_init() function.

Note

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

Parameters

in	*tp*	pointer to the thread_t structure

6.3.3.2.2.60 CH_CFG_THREAD_EXIT_HOOK

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

Value:

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

Threads finalization hook.

User finalization code added to the chThdExit() API.

Parameters

in	*tp*	pointer to the thread_t structure

6.3.3.2.2.61 CH_CFG_CONTEXT_SWITCH_HOOK

```
#define CH_CFG_CONTEXT_SWITCH_HOOK(
              ntp,
              otp)
```

Value:

```
  {                                 \
  /* Context switch code here.*/                                    \
}
```

Context switch hook.

This hook is invoked just before switching between threads.

Parameters

in	*ntp*	thread being switched in
in	*otp*	thread being switched out

6.3.3.2.2.62 CH_CFG_IRQ_PROLOGUE_HOOK

```
#define CH_CFG_IRQ_PROLOGUE_HOOK()
```

Value:

```
  {                                       \
  /* IRQ prologue code here.*/                                      \
}
```

ISR enter hook.

6.3.3.2.2.63 CH_CFG_IRQ_EPILOGUE_HOOK

```
#define CH_CFG_IRQ_EPILOGUE_HOOK()
```

Value:

```
  {                                       \
  /* IRQ epilogue code here.*/                                      \
}
```

ISR exit hook.

6.3.3.2.2.64 CH_CFG_IDLE_ENTER_HOOK

```
#define CH_CFG_IDLE_ENTER_HOOK()
```

Value:

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

Idle thread enter hook.

Note

This hook is invoked within a critical zone, no OS functions should be invoked from here.

This macro can be used to activate a power saving mode.

6.3.3.2.2.65 CH_CFG_IDLE_LEAVE_HOOK

```
#define CH_CFG_IDLE_LEAVE_HOOK()
```

Value:

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

Idle thread leave hook.

Note

This hook is invoked within a critical zone, no OS functions should be invoked from here.

This macro can be used to deactivate a power saving mode.

6.3.3.2.2.66 CH_CFG_IDLE_LOOP_HOOK

```
#define CH_CFG_IDLE_LOOP_HOOK()
```

Value:

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

Idle Loop hook.

This hook is continuously invoked by the idle thread loop.

6.3.3.2.2.67 CH_CFG_SYSTEM_TICK_HOOK

```
#define CH_CFG_SYSTEM_TICK_HOOK()
```

Value:

```
  {                                       \
  /* System tick event code here.*/                            \
}
```

System tick event hook.

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

6.3.3.2.2.68 CH_CFG_SYSTEM_HALT_HOOK

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

Value:

```
  {                                   \
  /* System halt code here.*/                                  \
}
```

System halt hook.

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

6.3.3.2.2.69 CH_CFG_TRACE_HOOK

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

Value:

```
  {                                       \
  /* Trace code here.*/                                        \
}
```

Trace hook.

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

6.3.3.2.2.70 CH_CFG_RUNTIME_FAULTS_HOOK

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

Value:

```
  {                                   \
  /* Faults handling code here.*/                              \
}
```

Runtime Faults Collection Unit hook.

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

6.3.3.3 Checks

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

6.3.3.4 Restrictions

6.3.3.4.1 Detailed Description

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

Macros

- #define CH_DBG_STATISTICS FALSE
- #define CH_DBG_TRACE_MASK CH_DBG_TRACE_MASK_DISABLED
- #define CH_CFG_ST_TIMEDELTA 0
- #define CH_CFG_USE_TM FALSE
- #define CH_CFG_USE_MUTEXES FALSE
- #define CH_CFG_USE_CONDVARS FALSE
- #define CH_CFG_USE_DYNAMIC FALSE

6.3.3.4.2 Macro Definition Documentation

6.3.3.4.2.1 CH_DBG_STATISTICS

```
#define CH_DBG_STATISTICS FALSE
```

6.3.3.4.2.2 CH_DBG_TRACE_MASK

```
#define CH_DBG_TRACE_MASK CH_DBG_TRACE_MASK_DISABLED
```

6.3.3.4.2.3 CH_CFG_ST_TIMEDELTA

```
#define CH_CFG_ST_TIMEDELTA 0
```

6.3.3.4.2.4 CH_CFG_USE_TM

```
#define CH_CFG_USE_TM FALSE
```

6.3.3.4.2.5 CH_CFG_USE_MUTEXES

```
#define CH_CFG_USE_MUTEXES FALSE
```

6.3.3.4.2.6 CH_CFG_USE_CONDVARS

```
#define CH_CFG_USE_CONDVARS FALSE
```

6.3.3.4.2.7 CH_CFG_USE_DYNAMIC

```
#define CH_CFG_USE_DYNAMIC FALSE
```

6.3.4 System

6.3.4.1 Detailed Description

Topics

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

6.3.4.2 Port Interface

6.3.4.2.1 Detailed Description

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

Note

This module does not export any functionality.

Macros

- #define PORT_CORES_NUMBER 1
- #define CH_PORT_SUPPORTS_RECURSIVE_LOCKS FALSE

6.3.4.2.2 Macro Definition Documentation

6.3.4.2.2.1 PORT_CORES_NUMBER

```
#define PORT_CORES_NUMBER 1
```

6.3.4.2.2.2 CH_PORT_SUPPORTS_RECURSIVE_LOCKS

```
#define CH_PORT_SUPPORTS_RECURSIVE_LOCKS FALSE
```

6.3.4.3 OS Types and Structures

6.3.4.3.1 Detailed Description

Kernel types

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

Data Structures

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

Macros

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

Typedefs

- typedef unsigned core_id_t
  Type of a core identifier.
- typedef struct ch_thread thread_t
  Type of a thread structure.
- typedef struct ch_os_instance os_instance_t
  Type of an OS instance structure.
- typedef struct ch_virtual_timer virtual_timer_t
  Type of a Virtual Timer.
- typedef void(∗ vtfunc_t) (virtual_timer_t ∗vtp, void ∗p)
  Type of a Virtual Timer callback function.
- typedef struct ch_virtual_timers_list virtual_timers_list_t
  Type of virtual timers list header.
- typedef struct ch_registry registry_t
  Type of a registry structure.
- typedef thread_t ∗ thread_reference_t
  Type of a thread reference.
- typedef struct ch_threads_queue threads_queue_t
  Type of a threads queue.
- typedef struct ch_ready_list ready_list_t
  Type of a ready list header.
- typedef struct ch_os_instance_config os_instance_config_t
  Type of an system instance configuration.
- typedef struct ch_system ch_system_t
  Type of system data structure.

Enumerations

- enum system_state_t { ch_sys_uninit = 0 , ch_sys_initializing = 1 , ch_sys_running = 2 , ch_sys_halted = 3 }
  Global state of the operating system.

Functions

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

6.3.4.3.2 Macro Definition Documentation

6.3.4.3.2.1 __CH_STRINGIFY

```
#define __CH_STRINGIFY(
              a)
```

Value:

```
#a
```

Utility to make the parameter a quoted string.

Parameters

in	*a*	literal to be string-ified

6.3.4.3.2.2 __CH_OFFSETOF

```
#define __CH_OFFSETOF(
              st,
              m)
```

Value:

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

Structure field offset utility.

Parameters

in	*st*	structured type name
in	*m*	field name in the structured type

Returns

The offset of the field in the structured type.

6.3.4.3.2.3 __CH_USED

```
#define __CH_USED(
              x)
```

Value:

```
(void)(x)
```

Marks an expression result as used.

Parameters

in	*x*	a valid expression

6.3.4.3.2.4 likely

```
#define likely(
              x)
```

Value:

```
PORT_LIKELY(x)
```

Marks a boolean expression as likely true.

Note

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

Parameters

in	*x*	a valid expression

6.3.4.3.2.5 unlikely

```
#define unlikely(
                x)
```

Value:

```
PORT_UNLIKELY(x)
```

Marks a boolean expression as likely false.

Note

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

Parameters

in	*x*	a valid expression

6.3.4.3.2.6 threadref

```
#define threadref(
                p)
```

Value:

```
((thread_t *)(void *)(p))
```

Safe cast of a queue pointer to a thread pointer.

Note

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

Parameters

in	*p*	pointer to a queue/list structure

Returns

The pointer to the thread containing the queue/list element.

6.3.4.3.3 Typedef Documentation

6.3.4.3.3.1 rtcnt_t

```
typedef port_rtcnt_t rtcnt_t
```

Realtime counter.

6.3.4.3.3.2 rttime_t

```
typedef port_rttime_t rttime_t
```

Realtime accumulator.

6.3.4.3.3.3 syssts_t

```
typedef port_syssts_t syssts_t
```

System status word.

6.3.4.3.3.4 stkalign_t

```
typedef port_stkalign_t stkalign_t
```

Stack alignment type.

6.3.4.3.3.5 tmode_t

```
typedef uint8_t tmode_t
```

Thread flags.

6.3.4.3.3.6 tstate_t

```
typedef uint8_t tstate_t
```

Thread state.

6.3.4.3.3.7 trefs_t

```
typedef uint8_t trefs_t
```

Thread references counter.

6.3.4.3.3.8 tslices_t

```
typedef uint8_t tslices_t
```

Thread time slices counter.

6.3.4.3.3.9 tprio_t

```
typedef uint32_t tprio_t
```

Thread priority.

6.3.4.3.3.10 msg_t

```
typedef int32_t msg_t
```

Inter-thread message.

6.3.4.3.3.11 eventid_t

```
typedef int32_t eventid_t
```

Numeric event identifier.

6.3.4.3.3.12 eventmask_t

```
typedef uint32_t eventmask_t
```

Mask of event identifiers.

6.3.4.3.3.13 eventflags_t

```
typedef uint32_t eventflags_t
```

Mask of event flags.

6.3.4.3.3.14 cnt_t

```
typedef int32_t cnt_t
```

Generic signed counter.

6.3.4.3.3.15 ucnt_t

```
typedef uint32_t ucnt_t
```

Generic unsigned counter.

6.3.4.3.3.16 core_id_t

```
typedef unsigned core_id_t
```

Type of a core identifier.

Note

Core identifiers have ranges from 0 to `PORT_CORES_NUMBER` - 1.

6.3.4.3.3.17 thread_t

```
typedef struct ch_thread thread_t
```

Type of a thread structure.

6.3.4.3.3.18 os_instance_t

```
typedef struct ch_os_instance os_instance_t
```

Type of an OS instance structure.

6.3.4.3.3.19 virtual_timer_t

```
typedef struct ch_virtual_timer virtual_timer_t
```

Type of a Virtual Timer.

6.3.4.3.3.20 vtfunc_t

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

Type of a Virtual Timer callback function.

Parameters

in	*vtp*	pointer to the `virtual_timer_t` calling this callback
in	*p*	optional argument to the callback

Returns

The interval to be reloaded into the timer or zero.

Return values

0	if the timer must not be reloaded.

6.3.4.3.3.21 virtual_timers_list_t

```
typedef struct ch_virtual_timers_list virtual_timers_list_t
```

Type of virtual timers list header.

Note

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

6.3.4.3.3.22 registry_t

```
typedef struct ch_registry registry_t
```

Type of a registry structure.

6.3.4.3.3.23 thread_reference_t

```
typedef thread_t* thread_reference_t
```

Type of a thread reference.

6.3.4.3.3.24 threads_queue_t

```
typedef struct ch_threads_queue threads_queue_t
```

Type of a threads queue.

6.3.4.3.3.25 ready_list_t

```
typedef struct ch_ready_list ready_list_t
```

Type of a ready list header.

6.3.4.3.3.26 os_instance_config_t

```
typedef struct ch_os_instance_config os_instance_config_t
```

Type of an system instance configuration.

6.3.4.3.3.27 ch_system_t

```
typedef struct ch_system ch_system_t
```

Type of system data structure.

6.3.4.3.4 Enumeration Type Documentation

6.3.4.3.4.1 system_state_t

```
enum system_state_t
```

Global state of the operating system.

Enumerator

ch_sys_uninit	
ch_sys_initializing	
ch_sys_running	
ch_sys_halted	

6.3.4.3.5 Function Documentation

6.3.4.3.5.1 chSysHalt()

```
void chSysHalt (
            const char * reason)
```

Halts the system.

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

Note

Can be invoked from any system state.

Parameters

in	*reason*	pointer to an error string

Function Class:

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

6.3.4.4 OS Instances

6.3.4.4.1 Detailed Description

OS instances management.

Macros

- #define __instance_get_currthread(oip)

  Current thread pointer get macro.
- #define __instance_set_currthread(oip, tp)

  Current thread pointer set macro.

Functions

- static void __idle_thread (void *p)

  This function implements the idle thread infinite loop.
- void chInstanceObjectInit (os_instance_t *oip, const os_instance_config_t *oicp)

  Initializes a system instance.

6.3.4.4.2 Macro Definition Documentation

6.3.4.4.2.1 __instance_get_currthread

```
#define __instance_get_currthread(
              oip)
```

Value:

```
(oip)->rlist.current
```

Current thread pointer get macro.

Note

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

6.3.4.4.2.2 __instance_set_currthread

```
#define __instance_set_currthread(
              oip,
              tp)
```

Value:

```
(oip)->rlist.current = (tp)
```

Current thread pointer set macro.

6.3.4.4.3 Function Documentation

6.3.4.4.3.1 __idle_thread()

```
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

6.3.4.4.3.2 chInstanceObjectInit()

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

Initializes a system instance.

Note

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

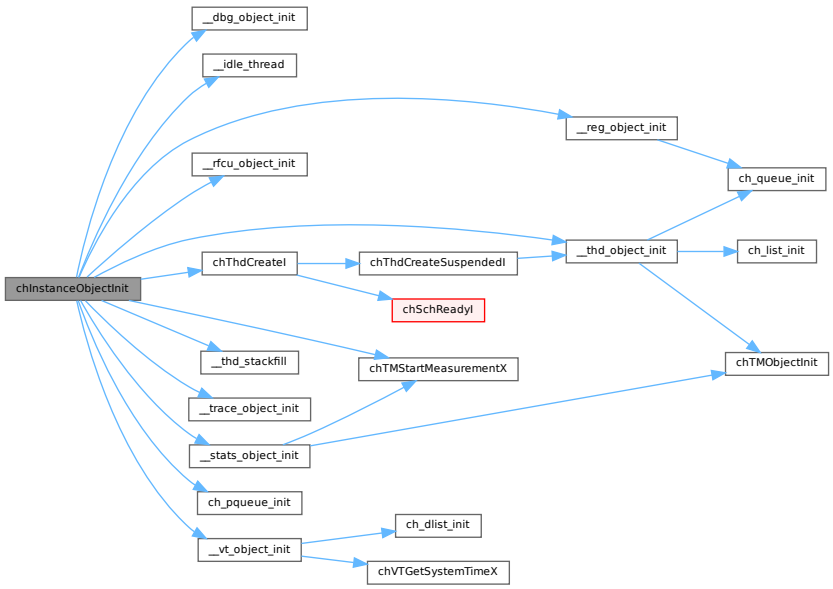
Parameters

out	*oip*	pointer to the os_instance_t structure
in	*oicp*	pointer to the os_instance_config_t structure

Function Class:

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

Here is the call graph for this function:



6.3.4.5 Runtime Faults Collection Unit

6.3.4.5.1 Detailed Description

Runtime Faults Collection Unit service.

Predefined Faults

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

Data Structures

- struct ch_rfcu
  Type of an RFCU structure.

Macros

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

Typedefs

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

Functions

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

6.3.4.5.2 Macro Definition Documentation

6.3.4.5.2.1 CH_RFCU_VT_INSUFFICIENT_DELTA

```
#define CH_RFCU_VT_INSUFFICIENT_DELTA 1U
```

6.3.4.5.2.2 CH_RFCU_VT_SKIPPED_DEADLINE

```
#define CH_RFCU_VT_SKIPPED_DEADLINE 2U
```

6.3.4.5.2.3 CH_RFCU_ALL_FAULTS

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

Mask of all faults.

6.3.4.5.3 Typedef Documentation

6.3.4.5.3.1 rfcu_mask_t

```
typedef uint32_t rfcu_mask_t
```

Type of a faults mask.

6.3.4.5.3.2 rfcu_t

```
typedef struct ch_rfcu rfcu_t
```

Type of an RFCU structure.

6.3.4.5.4 Function Documentation

6.3.4.5.4.1 chRFCUCollectFaultsI()

```
void chRFCUCollectFaultsI (
             rfcu_mask_t mask)
```

Adds fault flags to the current mask.

Parameters

in	*mask*	fault flags to be added

6.3.4.5.4.2 chRFCUGetAndClearFaultsI()

```
rfcu_mask_t chRFCUGetAndClearFaultsI (
             rfcu_mask_t mask)
```

Returns the current faults mask clearing it.

Parameters

in	*mask*	mask of faults to be read and cleared

Returns

The current faults mask.

Return values

0	if no faults were collected since last call to this function.

6.3.4.5.4.3 __rfcu_object_init()

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

Runtime Faults Collection Unit initialization.

Note

Internal use only.

Parameters

out	*rfcup*	pointer to the rfcu_t structure

Function Class:

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

6.3.4.6 Lists and Queues

6.3.4.6.1 Detailed Description

Data Structures

- struct ch_list

  Structure representing a generic single link list header and element.
- struct ch_queue

  Structure representing a generic bidirectional linked list header and element.
- struct ch_priority_queue

  Structure representing a generic priority-ordered bidirectional linked list header and element.
- struct ch_delta_list

  Delta list element and header structure.

Macros

- #define __CH_QUEUE_DATA(name)

  Data part of a static queue object initializer.
- #define CH_QUEUE_DECL(name)

  Static queue object initializer.

Typedefs

- typedef struct ch_list ch_list_t

  Type of a generic single link list header and element.
- typedef struct ch_queue ch_queue_t

  Type of a generic bidirectional linked list header and element.
- typedef struct ch_priority_queue ch_priority_queue_t

  Type of a generic priority-ordered bidirectional linked list header and element.
- typedef struct ch_delta_list ch_delta_list_t

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

Functions

- static void ch_list_init (ch_list_t *lp)

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

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

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

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

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

  Queue initialization.
- static bool ch_queue_isempty (const ch_queue_t *qp)

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

  Evaluates to `true` if the specified queue is not empty.
- static void ch_queue_insert (ch_queue_t *qp, ch_queue_t *p)

  Inserts an element into a queue.
- static ch_queue_t * ch_queue_fifo_remove (ch_queue_t *qp)

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

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

  Removes an element from a queue and returns it.
- static void ch_pqueue_init (ch_priority_queue_t *pqp)

  Priority queue initialization.
- static ch_priority_queue_t * ch_pqueue_remove_highest (ch_priority_queue_t *pqp)

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

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

  Inserts an element in the priority queue placing it ahead of its peers.
- static void ch_dlist_init (ch_delta_list_t *dlhp)

  Delta list initialization.
- static bool ch_dlist_isempty (ch_delta_list_t *dlhp)

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

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

- static bool ch_dlist_islast (ch_delta_list_t *dlhp, ch_delta_list_t *dlp)

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

  Fist element in the delta list check.
- static void ch_dlist_insert_after (ch_delta_list_t *dlhp, ch_delta_list_t *dlp, sysinterval_t delta)

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

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

  Inserts an element in a delta list.
- static ch_delta_list_t * ch_dlist_remove_first (ch_delta_list_t *dlhp)

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

  Dequeues an element from the delta list.

6.3.4.6.2 Macro Definition Documentation

6.3.4.6.2.1 __CH_QUEUE_DATA

```
#define __CH_QUEUE_DATA(
              name)
```

Value:

```
{(ch_queue_t *)&name, (ch_queue_t *)&name}
```

Data part of a static queue object initializer.

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

Parameters

in	*name*	the name of the queue variable

6.3.4.6.2.2 CH_QUEUE_DECL

```
#define CH_QUEUE_DECL(
              name)
```

Value:

```
    ch_queue_t name = __CH_QUEUE_DATA(name)
```

Static queue object initializer.

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

Parameters

in	*name*	the name of the queue variable

6.3.4.6.3 Typedef Documentation

6.3.4.6.3.1 ch_list_t

```
typedef struct ch_list ch_list_t
```

Type of a generic single link list header and element.

6.3.4.6.3.2 ch_queue_t

```
typedef struct ch_queue ch_queue_t
```

Type of a generic bidirectional linked list header and element.

6.3.4.6.3.3 ch_priority_queue_t

```
typedef struct ch_priority_queue ch_priority_queue_t
```

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

6.3.4.6.3.4 ch_delta_list_t

```
typedef struct ch_delta_list ch_delta_list_t
```

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

6.3.4.6.4 Function Documentation

6.3.4.6.4.1 ch_list_init()

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

List initialization.

Parameters

out	*lp*	pointer to the list header

Function Class:

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

6.3.4.6.4.2 ch_list_isempty()

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

Evaluates to `true` if the specified list is empty.

Parameters

in	*lp*	pointer to the list header

Returns

The status of the list.

Function Class:

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

6.3.4.6.4.3 ch_list_notempty()

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

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

Parameters

in	*lp*	pointer to the list header

Returns

The status of the list.

Function Class:

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

6.3.4.6.4.4 ch_list_link()

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

Pushes an element on top of a stack list.

Parameters

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

Function Class:

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

6.3.4.6.4.5 ch_list_unlink()

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

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

Precondition

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

Parameters

in	*lp*	the pointer to the list header

Returns

The removed element pointer.

Function Class:

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

6.3.4.6.4.6 ch_queue_init()

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

Queue initialization.

Parameters

out	*qp*	pointer to the queue header

Function Class:

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

6.3.4.6.4.7 ch_queue_isempty()

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

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

Parameters

in	*qp*	pointer to the queue header

Returns

The status of the queue.

Function Class:

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

6.3.4.6.4.8 ch_queue_notempty()

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

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

Parameters

in	*qp*	pointer to the queue header

Returns

The status of the queue.

Function Class:

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

6.3.4.6.4.9 ch_queue_insert()

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

Inserts an element into a queue.

Parameters

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

Function Class:

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

6.3.4.6.4.10 ch_queue_fifo_remove()

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

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

Note

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

Parameters

in	*qp*	the pointer to the queue list header

Returns

The removed element pointer.

Function Class:

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

6.3.4.6.4.11 ch_queue_lifo_remove()

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

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

Note

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

Parameters

in	*qp*	the pointer to the queue list header

Returns

The removed element pointer.

Function Class:

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

6.3.4.6.4.12 ch_queue_dequeue()

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

Removes an element from a queue and returns it.

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

Parameters

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

Returns

The removed element pointer.

Function Class:

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

6.3.4.6.4.13 ch_pqueue_init()

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

Priority queue initialization.

Note

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

Parameters

out	*pqp*	pointer to the priority queue header

Function Class:

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

6.3.4.6.4.14 ch_pqueue_remove_highest()

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

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

Parameters

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

Returns

The removed element pointer.

Function Class:

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

6.3.4.6.4.15 ch_pqueue_insert_behind()

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

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

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

Parameters

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

Returns

The inserted element pointer.

Function Class:

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

6.3.4.6.4.16 ch_pqueue_insert_ahead()

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

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

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

Parameters

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

Returns

The inserted element pointer.

Function Class:

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

6.3.4.6.4.17 ch_dlist_init()

```
void ch_dlist_init (
             ch_delta_list_t * dlhp)  [inline], [static]
```

Delta list initialization.

Parameters

out	*dlhp*	pointer to the delta list header

Function Class:

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

6.3.4.6.4.18 ch_dlist_isempty()

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

Evaluates to `true` if the specified delta list is empty.

Parameters

in	*dlhp*	pointer to the delta list header

Returns

The status of the delta list.

Function Class:

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

6.3.4.6.4.19 ch_dlist_notempty()

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

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

Parameters

in	*dlhp*	pointer to the delta list header

Returns

The status of the delta list.

Function Class:

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

6.3.4.6.4.20 ch_dlist_islast()

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

Last element in the delta list check.

Parameters

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

Function Class:

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

6.3.4.6.4.21 ch_dlist_isfirst()

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

Fist element in the delta list check.

Parameters

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

Function Class:

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

6.3.4.6.4.22 ch_dlist_insert_after()

```
void ch_dlist_insert_after (
             ch_delta_list_t * dlhp,
             ch_delta_list_t * dlp,
             sysinterval_t delta)  [inline], [static]
```

Inserts an element after another header element.

Parameters

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

Function Class:

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

6.3.4.6.4.23 ch_dlist_insert_before()

```
void ch_dlist_insert_before (
             ch_delta_list_t * dlhp,
             ch_delta_list_t * dlp,
             sysinterval_t delta)  [inline], [static]
```

Inserts an element before another header element.

Parameters

in	*dlhp*	pointer to the delta list header element
in	*dlp*	element to be inserted before the header element
in	*delta*	delta of the element to be inserted

Function Class:

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

6.3.4.6.4.24 ch_dlist_insert()

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

Inserts an element in a delta list.

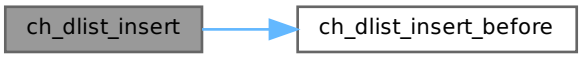
Parameters

in	*dlhp*	pointer to the delta list header element
in	*dlep*	element to be inserted before the header element
in	*delta*	delta of the element to be inserted

Function Class:

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

Here is the call graph for this function:



6.3.4.6.4.25 ch_dlist_remove_first()

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

Dequeues an element from the delta list.

Parameters

in	*dlhp*	pointer to the delta list header

Function Class:

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

6.3.4.6.4.26 ch_dlist_dequeue()

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

Dequeues an element from the delta list.

Parameters

in	*dlp*	pointer to the delta list element

Function Class:

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

6.3.4.7 Scheduler

6.3.4.7.1 Detailed Description

This module provides the default portable scheduler code.

Wakeup status codes

- #define MSG_OK (msg_t)0

  Normal wakeup message.
- #define MSG_TIMEOUT (msg_t)-1

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

  Wakeup caused by a reset condition.

Priority constants

- #define NOPRIO (tprio_t)0

  Ready list header priority.
- #define IDLEPRIO (tprio_t)1

  Idle priority.
- #define LOWPRIO (tprio_t)2

  Lowest priority.
- #define NORMALPRIO (tprio_t)128

  Normal priority.
- #define HIGHPRIO (tprio_t)255

  Highest priority.

Thread states

- #define CH_STATE_READY (tstate_t)0

  Waiting on the ready list.
- #define CH_STATE_CURRENT (tstate_t)1

  Currently running.
- #define CH_STATE_WTSTART (tstate_t)2

  Just created.
- #define CH_STATE_SUSPENDED (tstate_t)3

  Suspended state.
- #define CH_STATE_QUEUED (tstate_t)4

  On a queue.
- #define CH_STATE_WTSEM (tstate_t)5

  On a semaphore.
- #define CH_STATE_WTMTX (tstate_t)6

  On a mutex.
- #define CH_STATE_WTCOND (tstate_t)7

  On a cond.variable.
- #define CH_STATE_SLEEPING (tstate_t)8

  Sleeping.
- #define CH_STATE_WTEXIT (tstate_t)9

  Waiting a thread.
- #define CH_STATE_WTOREVT (tstate_t)10

  One event.
- #define CH_STATE_WTANDEVT (tstate_t)11

  Several events.
- #define CH_STATE_SNDMSGQ (tstate_t)12

  Sending a message, in queue.

- #define CH_STATE_SNDMSG (tstate_t)13

  Sent a message, waiting answer.
- #define CH_STATE_WTMSG (tstate_t)14

  Waiting for a message.
- #define CH_STATE_FINAL (tstate_t)15

  Thread terminated.
- #define CH_STATE_NAMES

  Thread states as array of strings.

Thread flags and attributes

- #define CH_FLAG_MODE_MASK (tmode_t)3U

  Thread memory mode mask.
- #define CH_FLAG_MODE_STATIC (tmode_t)0U

  Static thread.
- #define CH_FLAG_MODE_HEAP (tmode_t)1U

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

  Thread allocated from a Memory Pool.
- #define CH_FLAG_TERMINATE (tmode_t)4U

  Termination requested flag.

Macros

- #define firstprio(rlp)

  Returns the priority of the first thread on the given ready list.
- #define __sch_get_currthread()

  Current thread pointer get macro.

Functions

- static thread_t * __sch_ready_behind (thread_t *tp)

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

  Inserts a thread in the Ready List placing it ahead its peers.
- static void __sch_reschedule_behind (void)

  Switches to the first thread on the runnable queue.
- static void __sch_reschedule_ahead (void)

  Switches to the first thread on the runnable queue.
- static void __sch_wakeup (virtual_timer_t *vtp, void *p)
- void ch_sch_prio_insert (ch_queue_t *qp, ch_queue_t *tp)

  Inserts a thread into a priority ordered queue.
- thread_t * chSchReadyI (thread_t *tp)

  Inserts a thread in the Ready List placing it behind its peers.
- void chSchGoSleepS (tstate_t newstate)

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

  Puts the current thread to sleep into the specified state with timeout specification.
- void chSchWakeupS (thread_t *ntp, msg_t msg)

Wakes up a thread.

- void chSchRescheduleS (void)

  Performs a reschedule if a higher priority thread is runnable.
- bool chSchIsPreemptionRequired (void)

  Evaluates if preemption is required.
- void chSchDoPreemption (void)

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

  All-in-one preemption code.
- void chSchDoYieldS (void)

  Yields the time slot.
- thread_t * chSchSelectFirst (void)

  Makes runnable the fist thread in the ready list, does not reschedule internally.
- void chSchObjectInit (os_instance_t *oip, const os_instance_config_t *oicp)

6.3.4.7.2 Macro Definition Documentation

6.3.4.7.2.1 MSG_OK

```
#define MSG_OK (msg_t)0
```

Normal wakeup message.

6.3.4.7.2.2 MSG_TIMEOUT

```
#define MSG_TIMEOUT (msg_t)-1
```

Wakeup caused by a timeout condition.

6.3.4.7.2.3 MSG_RESET

```
#define MSG_RESET (msg_t)-2
```

Wakeup caused by a reset condition.

6.3.4.7.2.4 NOPRIO

```
#define NOPRIO (tprio_t)0
```

Ready list header priority.

6.3.4.7.2.5 IDLEPRIO

```
#define IDLEPRIO (tprio_t)1
```

Idle priority.

6.3.4.7.2.6 LOWPRIO

```
#define LOWPRIO (tprio_t)2
```

Lowest priority.

6.3.4.7.2.7 NORMALPRIO

```
#define NORMALPRIO (tprio_t)128
```

Normal priority.

6.3.4.7.2.8 HIGHPRIO

```
#define HIGHPRIO (tprio_t)255
```

Highest priority.

6.3.4.7.2.9 CH_STATE_READY

```
#define CH_STATE_READY (tstate_t)0
```

Waiting on the ready list.

6.3.4.7.2.10 CH_STATE_CURRENT

```
#define CH_STATE_CURRENT (tstate_t)1
```

Currently running.

6.3.4.7.2.11 CH_STATE_WTSTART

```
#define CH_STATE_WTSTART (tstate_t)2
```

Just created.

6.3.4.7.2.12 CH_STATE_SUSPENDED

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

Suspended state.

6.3.4.7.2.13 CH_STATE_QUEUED

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

On a queue.

6.3.4.7.2.14 CH_STATE_WTSEM

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

On a semaphore.

6.3.4.7.2.15 CH_STATE_WTMTX

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

On a mutex.

6.3.4.7.2.16 CH_STATE_WTCOND

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

On a cond.variable.

6.3.4.7.2.17 CH_STATE_SLEEPING

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

Sleeping.

6.3.4.7.2.18 CH_STATE_WTEXIT

```
#define CH_STATE_WTEXIT (tstate_t)9
```

Waiting a thread.

6.3.4.7.2.19 CH_STATE_WTOREVT

```
#define CH_STATE_WTOREVT (tstate_t)10
```

One event.

6.3.4.7.2.20 CH_STATE_WTANDEVT

```
#define CH_STATE_WTANDEVT (tstate_t)11
```

Several events.

6.3.4.7.2.21 CH_STATE_SNDMSGQ

```
#define CH_STATE_SNDMSGQ (tstate_t)12
```

Sending a message, in queue.

6.3.4.7.2.22 CH_STATE_SNDMSG

```
#define CH_STATE_SNDMSG (tstate_t)13
```

Sent a message, waiting answer.

6.3.4.7.2.23 CH_STATE_WTMSG

```
#define CH_STATE_WTMSG (tstate_t)14
```

Waiting for a message.

6.3.4.7.2.24 CH_STATE_FINAL

```
#define CH_STATE_FINAL (tstate_t)15
```

Thread terminated.

6.3.4.7.2.25 CH_STATE_NAMES

```
#define CH_STATE_NAMES
```

Value:

```
  "READY", "CURRENT", "WTSTART", "SUSPENDED", "QUEUED", "WTSEM", "WTMTX",  \
  "WTCOND", "SLEEPING", "WTEXIT", "WTOREVT", "WTANDEVT", "SNDMSGQ",        \
  "SNDMSG", "WTMSG", "FINAL"
```

Thread states as array of strings.

Each element in an array initialized with this macro can be indexed using the numeric thread state values.

6.3.4.7.2.26 CH_FLAG_MODE_MASK

```
#define CH_FLAG_MODE_MASK (tmode_t)3U
```

Thread memory mode mask.

6.3.4.7.2.27 CH_FLAG_MODE_STATIC

```
#define CH_FLAG_MODE_STATIC (tmode_t)0U
```

Static thread.

6.3.4.7.2.28 CH_FLAG_MODE_HEAP

```
#define CH_FLAG_MODE_HEAP (tmode_t)1U
```

Thread allocated from a Memory Heap.

6.3.4.7.2.29 CH_FLAG_MODE_MPOOL

```
#define CH_FLAG_MODE_MPOOL (tmode_t)2U
```

Thread allocated from a Memory Pool.

6.3.4.7.2.30 CH_FLAG_TERMINATE

```
#define CH_FLAG_TERMINATE (tmode_t)4U
```

Termination requested flag.

6.3.4.7.2.31 firstprio

```
#define firstprio(
              rlp)
```

Value:

```
((rlp)->next->prio)
```

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

Function Class:

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

6.3.4.7.2.32 __sch_get_currthread

```
#define __sch_get_currthread()
```

Value:

```
__instance_get_currthread(currcore)
```

Current thread pointer get macro.

Note

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

6.3.4.7.3 Function Documentation

6.3.4.7.3.1 __sch_ready_behind()

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

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

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

Precondition

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

Postcondition

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

Parameters

in	*tp*	the thread to be made ready

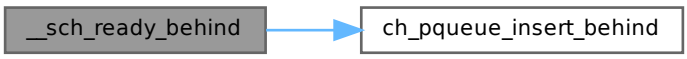
Returns

The thread pointer.

Function Class:

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

Here is the call graph for this function:



6.3.4.7.3.2 __sch_ready_ahead()

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

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

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

Precondition

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

Postcondition

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

Parameters

in	*tp*	the thread to be made ready

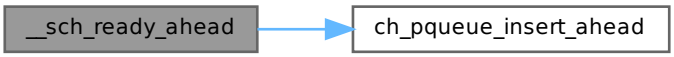
Returns

The thread pointer.

Function Class:

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

Here is the call graph for this function:



6.3.4.7.3.3 __sch_reschedule_behind()

```
void __sch_reschedule_behind (
             void )  [static]
```

Switches to the first thread on the runnable queue.

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

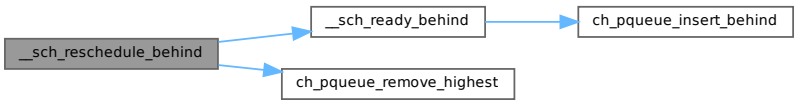
Note

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

Function Class:

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

Here is the call graph for this function:



6.3.4.7.3.4 __sch_reschedule_ahead()

```
void __sch_reschedule_ahead (
            void )  [static]
```

Switches to the first thread on the runnable queue.

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

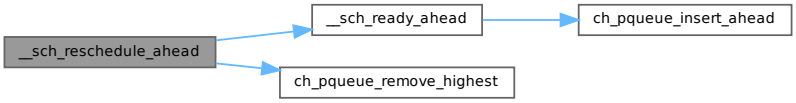
Note

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

Function Class:

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

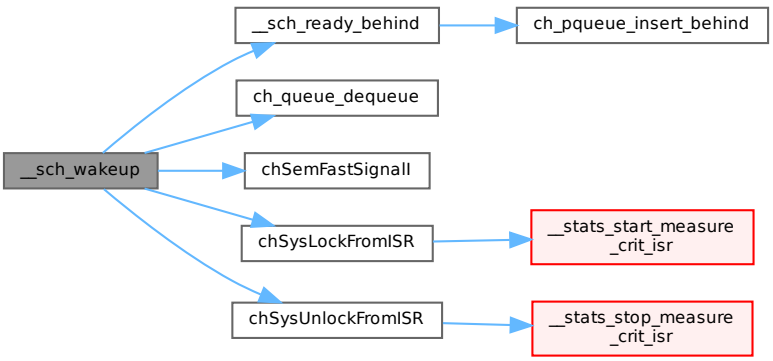
Here is the call graph for this function:



6.3.4.7.3.5 __sch_wakeup()

```
void __sch_wakeup (
            virtual_timer_t * vtp,
            void * p)  [static]
```

Here is the call graph for this function:



6.3.4.7.3.6 ch_sch_prio_insert()

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

Inserts a thread into a priority ordered queue.

Note

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

Parameters

in	*qp*	the pointer to the threads list header
in	*tp*	the pointer to the thread to be inserted in the list

Function Class:

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

6.3.4.7.3.7 chSchReadyI()

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

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

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

Precondition

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

Postcondition

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

Parameters

in	*tp*	the thread to be made ready

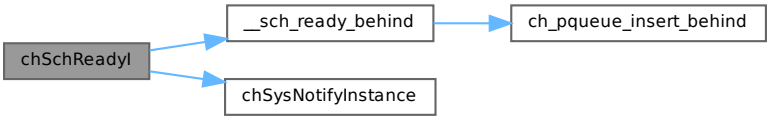
Returns

The thread pointer.

Function Class:

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

Here is the call graph for this function:



6.3.4.7.3.8 chSchGoSleepS()

```
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 `chschd.h`.

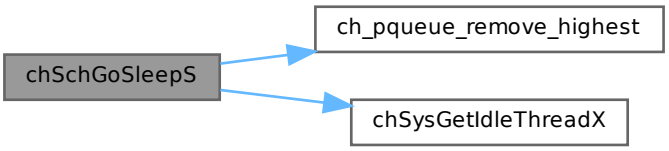
Parameters

`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:



6.3.4.7.3.9 chSchGoSleepTimeoutS()

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

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

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

Parameters

in	*newstate*	the new thread state
in	*timeout*	the number of ticks before the operation timeouts, the special values are handled as follow: • *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.

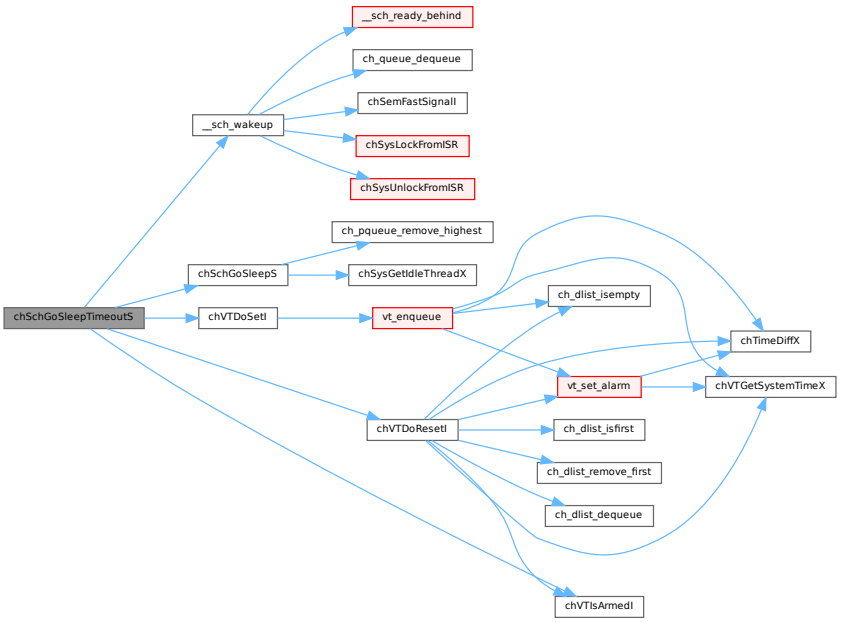
Return values

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:



6.3.4.7.3.10 chSchWakeupS()

```
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.

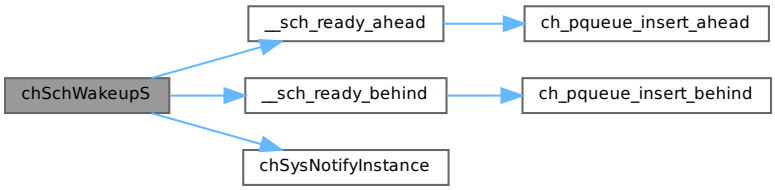
Parameters

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

Function Class:

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

Here is the call graph for this function:



6.3.4.7.3.11 chSchRescheduleS()

```
void chSchRescheduleS (
            void )
```

Performs a reschedule if a higher priority thread is runnable.

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

Note

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

Function Class:

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

Here is the call graph for this function:

6.3.4.7.3.12 chSchIsPreemptionRequired()

```
bool chSchIsPreemptionRequired (
            void )
```

Evaluates if preemption is required.

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

Note

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

Return values

true	if there is a thread that must go in running state immediately.
false	if preemption is not required.

Function Class:

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

6.3.4.7.3.13 chSchDoPreemption()

```
void chSchDoPreemption (
            void )
```

Switches to the first thread on the runnable queue.

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

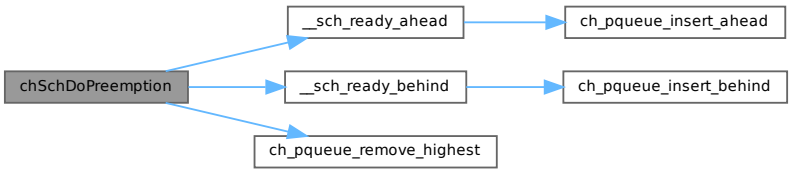
Note

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

Function Class:

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

Here is the call graph for this function:



6.3.4.7.3.14 chSchPreemption()

```
void chSchPreemption (
            void )
```

All-in-one preemption code.

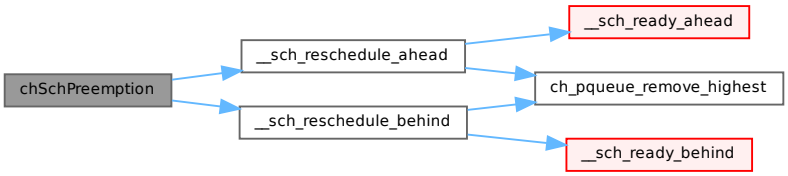
Note

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

Function Class:

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

Here is the call graph for this function:



6.3.4.7.3.15 chSchDoYieldS()

```
void chSchDoYieldS (
             void )
```

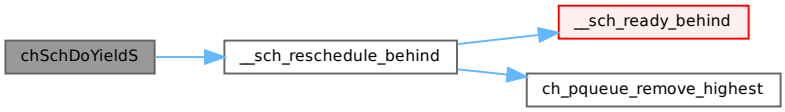
Yields the time slot.

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

Function Class:

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

Here is the call graph for this function:



6.3.4.7.3.16 chSchSelectFirst()

```
thread_t * chSchSelectFirst (
             void )
```

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

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

Note

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

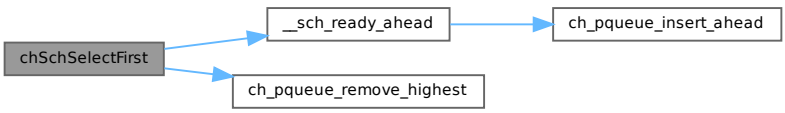
Returns

The pointer to the thread being switched in.

Function Class:

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

Here is the call graph for this function:



6.3.4.7.3.17 chSchObjectInit()

```
void chSchObjectInit (
             os_instance_t * oip,
             const os_instance_config_t * oicp)
```

6.3.5 Base Kernel Services

6.3.5.1 Detailed Description

Topics

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

6.3.5.2 System Management

6.3.5.2.1 Detailed Description

System related APIs and services:

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

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)
  Priority level validation macro.
- #define CH_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)
  Standard normal IRQ handler declaration.

Fast ISRs abstraction macros

- #define CH_FAST_IRQ_HANDLER(id)

  Standard fast IRQ handler declaration.

Time conversion utilities for the realtime counter

- #define S2RTC(freq, sec)

  Seconds to realtime counter.
- #define MS2RTC(freq, msec)

  Milliseconds to realtime counter.
- #define US2RTC(freq, usec)

  Microseconds to realtime counter.
- #define RTC2S(freq, n)

  Realtime counter cycles to seconds.
- #define RTC2MS(freq, n)

  Realtime counter cycles to milliseconds.
- #define RTC2US(freq, n)

  Realtime counter cycles to microseconds.

Macros

- #define CH_SYS_CORE0_MEMORY PORT_CORE0_BSS_SECTION

  Core zero memory affinity macro.
- #define CH_SYS_CORE1_MEMORY PORT_CORE1_BSS_SECTION

  Core one memory affinity macro.
- #define currcore ch_system.instances[port_get_core_id()]

  Access to current core's instance structure.
- #define chSysGetRealtimeCounterX()

  Returns the current value of the system real time counter.
- #define chSysSwitch(ntp, otp)

  Performs a context switch.

Functions

- static CH_SYS_CORE0_MEMORY THD_WORKING_AREA (ch_c0_idle_thread_wa, PORT_IDLE_↩ THREAD_STACK_SIZE)

  Working area for core 0 idle thread.
- static CH_SYS_CORE1_MEMORY THD_WORKING_AREA (ch_c1_idle_thread_wa, PORT_IDLE_↩ THREAD_STACK_SIZE)

  Working area for core 1 idle thread.
- void chSysWaitSystemState (system_state_t state)

  Waits for the system state to be equal to the specified one.
- void chSysInit (void)

  System initialization.
- void chSysHalt (const char *reason)

  Halts the system.
- thread_t * chSysGetIdleThreadX (void)

  Returns a pointer to the idle thread.

- bool chSysIntegrityCheckI (unsigned testmask)

  System integrity check.
- void chSysTimerHandlerI (void)

  Handles time ticks for round robin preemption and timer increments.
- syssts_t chSysGetStatusAndLockX (void)

  Returns the execution status and enters a critical zone.
- void chSysRestoreStatusX (syssts_t sts)

  Restores the specified execution status and leaves a critical zone.
- bool chSysIsCounterWithinX (rtcnt_t cnt, rtcnt_t start, rtcnt_t end)

  Realtime window test.
- void chSysPolledDelayX (rtcnt_t cycles)

  Polled delay.
- static void chSysDisable (void)

  Raises the system interrupt priority mask to the maximum level.
- static void chSysSuspend (void)

  Raises the system interrupt priority mask to system level.
- static void chSysEnable (void)

  Lowers the system interrupt priority mask to user level.
- static void chSysLock (void)

  Enters the kernel lock state.
- static void chSysUnlock (void)

  Leaves the kernel lock state.
- static void chSysLockFromISR (void)

  Enters the kernel lock state from within an interrupt handler.
- static void chSysUnlockFromISR (void)

  Leaves the kernel lock state from within an interrupt handler.
- static void chSysUnconditionalLock (void)

  Unconditionally enters the kernel lock state.
- static void chSysUnconditionalUnlock (void)

  Unconditionally leaves the kernel lock state.
- static void chSysNotifyInstance (os_instance_t *oip)

  Notifies an OS instance to check for reschedule.

Variables

- ch_system_t ch_system

  System root object.
- CH_SYS_CORE0_MEMORY os_instance_t ch0

  Core 0 OS instance.
- stkalign_t __main_thread_stack_base__
- stkalign_t __main_thread_stack_end__
- const os_instance_config_t ch_core0_cfg

  Core 0 OS instance configuration.
- CH_SYS_CORE1_MEMORY os_instance_t ch1

  Core 1 OS instance.
- stkalign_t __c1_main_thread_stack_base__
- stkalign_t __c1_main_thread_stack_end__
- const os_instance_config_t ch_core1_cfg

  Core 1 OS instance configuration.

6.3.5.2.2 Macro Definition Documentation

6.3.5.2.2.1 CH_INTEGRITY_RLIST

```
#define CH_INTEGRITY_RLIST 1U
```

6.3.5.2.2.2 CH_INTEGRITY_VTLIST

```
#define CH_INTEGRITY_VTLIST 2U
```

6.3.5.2.2.3 CH_INTEGRITY_REGISTRY

```
#define CH_INTEGRITY_REGISTRY 4U
```

6.3.5.2.2.4 CH_INTEGRITY_PORT

```
#define CH_INTEGRITY_PORT 8U
```

6.3.5.2.2.5 CH_SYS_CORE0_MEMORY

```
#define CH_SYS_CORE0_MEMORY PORT_CORE0_BSS_SECTION
```

Core zero memory affinity macro.

Note

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

6.3.5.2.2.6 CH_SYS_CORE1_MEMORY

```
#define CH_SYS_CORE1_MEMORY PORT_CORE1_BSS_SECTION
```

Core one memory affinity macro.

Note

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

6.3.5.2.2.7 currcore

```
#define currcore ch_system.instances[port_get_core_id()]
```

Access to current core's instance structure.

6.3.5.2.2.8 CH_IRQ_IS_VALID_PRIORITY

```
#define CH_IRQ_IS_VALID_PRIORITY(
              prio)
```

Value:

```
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.

6.3.5.2.2.9 CH_IRQ_IS_VALID_KERNEL_PRIORITY

```
#define CH_IRQ_IS_VALID_KERNEL_PRIORITY(
              prio)
```

Value:

```
PORT_IRQ_IS_VALID_KERNEL_PRIORITY(prio)
```

Priority level validation macro.

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

Parameters

in	*prio*	the priority level

Returns

Priority range result.

Return values

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

6.3.5.2.2.10 CH_IRQ_PROLOGUE

```
#define CH_IRQ_PROLOGUE()
```

Value:

```
PORT_IRQ_PROLOGUE();                                                \
CH_CFG_IRQ_PROLOGUE_HOOK();                                         \
__stats_increase_irq();                                             \
__trace_isr_enter(__func__);                                        \
__dbg_check_enter_isr()
```

IRQ handler enter code.

Note

Usually IRQ handlers functions are also declared naked.

On some architectures this macro can be empty.

Function Class:

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

6.3.5.2.2.11 CH_IRQ_EPILOGUE

```
#define CH_IRQ_EPILOGUE()
```

Value:

```
__dbg_check_leave_isr();                                            \
__trace_isr_leave(__func__);                                        \
CH_CFG_IRQ_EPILOGUE_HOOK();                                         \
PORT_IRQ_EPILOGUE()
```

IRQ handler exit code.

Note

Usually IRQ handlers function are also declared naked.

This macro usually performs the final reschedule by using `chSchIsPreemptionRequired()` and `chSchDoReschedule()`.

Function Class:

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

6.3.5.2.2.12 CH_IRQ_HANDLER

```
#define CH_IRQ_HANDLER(
              id)
```

Value:

```
PORT_IRQ_HANDLER(id)
```

Standard normal IRQ handler declaration.

Note

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

Function Class:

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

6.3.5.2.2.13 CH_FAST_IRQ_HANDLER

```
#define CH_FAST_IRQ_HANDLER(
              id)
```

Value:

```
PORT_FAST_IRQ_HANDLER(id)
```

Standard fast IRQ handler declaration.

Note

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

Not all architectures support fast interrupts.

Function Class:

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

6.3.5.2.2.14 S2RTC

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

Value:

```
((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.

6.3.5.2.2.15 MS2RTC

```
#define MS2RTC(
              freq,
              msec)
```

Value:

```
(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.

6.3.5.2.2.16 US2RTC

```
#define US2RTC(
               freq,
               usec)
```

Value:

```
(rtcnt_t)((((freq) + 999999UL) / 1000000UL) * (usec))
```

Microseconds to realtime counter.

Converts from microseconds to realtime counter cycles.

Note

The result is rounded upward to the next microsecond boundary.

The macro assumes that `freq` >= `1000000`.

Parameters

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

Returns

The number of cycles.

Function Class:

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

6.3.5.2.2.17 RTC2S

```
#define RTC2S(
               freq,
               n)
```

Value:

```
((((n) - 1UL) / (freq)) + 1UL)
```

Realtime counter cycles to seconds.

Converts from realtime counter cycles number to seconds.

Note

The result is rounded up to the next second boundary.

The macro assumes that `freq` >= `1`.

Parameters

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

Returns

The number of seconds.

Function Class:

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

6.3.5.2.2.18 RTC2MS

```
#define RTC2MS(
               freq,
               n)
```

Value:

```
((((n) - 1UL) / ((freq) / 1000UL)) + 1UL)
```

Realtime counter cycles to milliseconds.

Converts from realtime counter cycles number to milliseconds.

Note

The result is rounded up to the next millisecond boundary.

The macro assumes that `freq` >= `1000`.

Parameters

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

Returns

The number of milliseconds.

Function Class:

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

6.3.5.2.2.19 RTC2US

```
#define RTC2US(
               freq,
               n)
```

Value:

```
((((n) - 1UL) / ((freq) / 1000000UL)) + 1UL)
```

Realtime counter cycles to microseconds.

Converts from realtime counter cycles number to microseconds.

Note

The result is rounded up to the next microsecond boundary.

The macro assumes that `freq` >= `1000000`.

Parameters

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

Returns

The number of microseconds.

Function Class:

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

6.3.5.2.2.20 chSysGetRealtimeCounterX

```
#define chSysGetRealtimeCounterX()
```

Value:

```
(rtcnt_t)port_rt_get_counter_value()
```

Returns the current value of the system real time counter.

Note

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

Returns

The value of the system realtime counter of type rtcnt_t.

Function Class:

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

6.3.5.2.2.21 chSysSwitch

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

Value:

```
                                                                {
     \
                                                                \
  __trace_switch(ntp, otp);                                     \
  __stats_ctxswc(ntp, otp);                                     \
  CH_CFG_CONTEXT_SWITCH_HOOK(ntp, otp);                         \
  port_switch(ntp, otp);                                        \
}
```

Performs a context switch.

Note

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

Parameters

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

Function Class:

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

6.3.5.2.3 Function Documentation

6.3.5.2.3.1 THD_WORKING_AREA() [1/2]

```
CH_SYS_CORE0_MEMORY THD_WORKING_AREA (
             ch_c0_idle_thread_wa ,
             PORT_IDLE_THREAD_STACK_SIZE )  [static]
```

Working area for core 0 idle thread.

6.3.5.2.3.2 THD_WORKING_AREA() [2/2]

```
CH_SYS_CORE1_MEMORY THD_WORKING_AREA (
             ch_c1_idle_thread_wa ,
             PORT_IDLE_THREAD_STACK_SIZE )  [static]
```

Working area for core 1 idle thread.

6.3.5.2.3.3 chSysWaitSystemState()

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

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

Note

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

Function Class:

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

6.3.5.2.3.4 chSysInit()

```
void chSysInit (
            void )
```

System initialization.

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

Precondition

Interrupts must disabled before invoking this function.

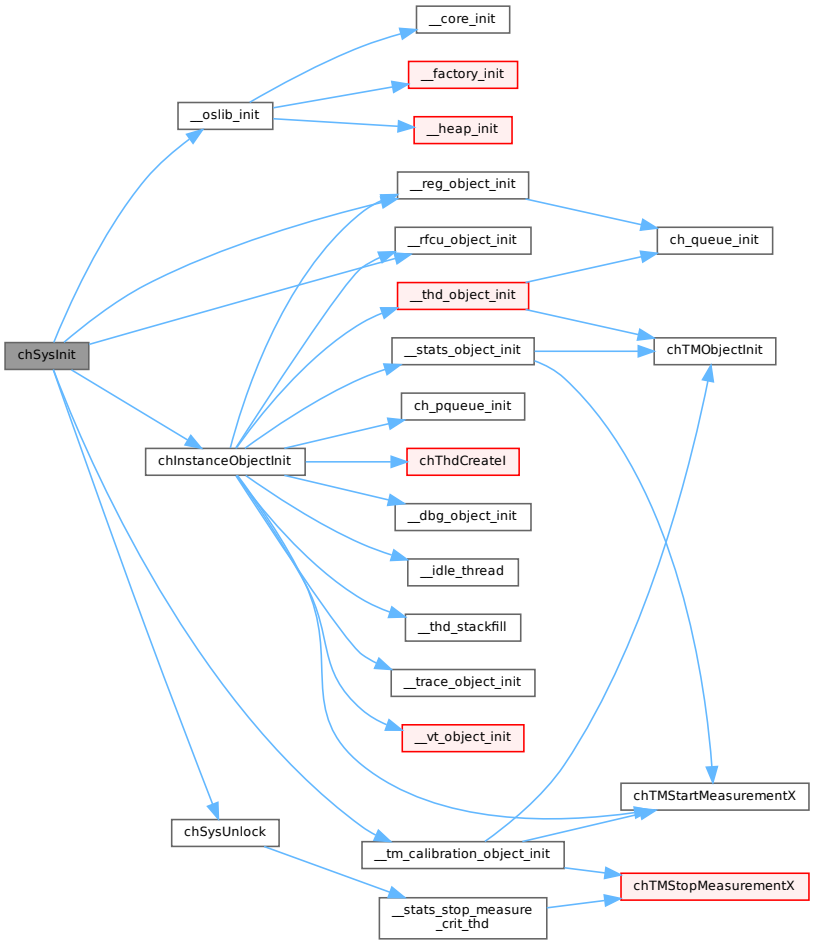
Postcondition

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

Function Class:

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

Here is the call graph for this function:



6.3.5.2.3.5 chSysHalt()

```
void chSysHalt (
            const char * reason)
```

Halts the system.

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

Note

Can be invoked from any system state.

Parameters

in	*reason*	pointer to an error string

Function Class:

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

6.3.5.2.3.6 chSysGetIdleThreadX()

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

Returns a pointer to the idle thread.

Note

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

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

Returns

Pointer to the idle thread.

Function Class:

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

6.3.5.2.3.7 chSysIntegrityCheckI()

```
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.

6.3.5.2.3.8 chSysTimerHandlerI()

```
void chSysTimerHandlerI (
              void )
```

Handles time ticks for round robin preemption and timer increments.

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

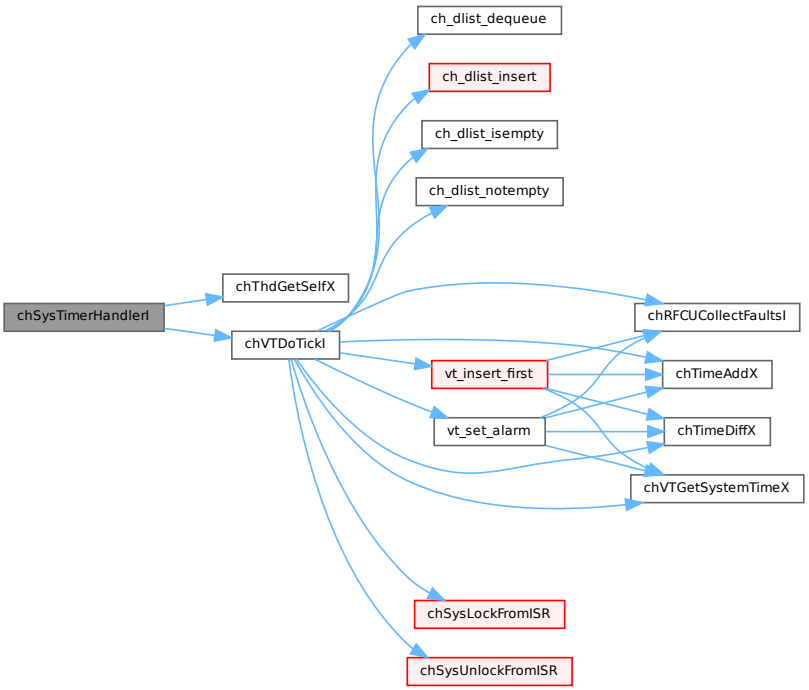
Note

The frequency of the timer determines the system tick granularity and, together with the CH_CFG_TIME_↩QUANTUM macro, the round robin interval.

Function Class:

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

Here is the call graph for this function:



6.3.5.2.3.9 chSysGetStatusAndLockX()

```
syssts_t chSysGetStatusAndLockX (
            void )
```

Returns the execution status and enters a critical zone.

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

Postcondition

The system is in a critical zone.

Note

This function is only available if the underlying port supports `port_get_lock_status()` and `port_is_locked()`.

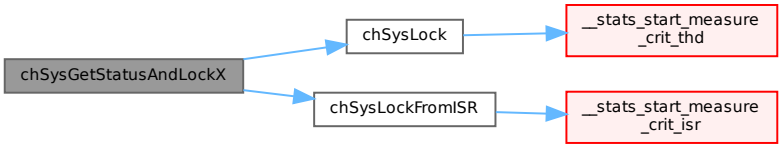
Returns

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

Function Class:

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

Here is the call graph for this function:



6.3.5.2.3.10 chSysRestoreStatusX()

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

Restores the specified execution status and leaves a critical zone.

Note

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

This function is only available if the underlying port supports `port_get_lock_status()` and `port_is_locked()`.

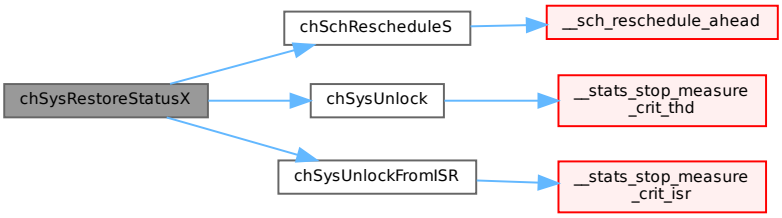
Parameters

`in`	*sts*	the system status to be restored.

Function Class:

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

Here is the call graph for this function:



6.3.5.2.3.11 chSysIsCounterWithinX()

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

Realtime window test.

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

Note

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

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

Parameters

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

Return values

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

Function Class:

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

6.3.5.2.3.12 chSysPolledDelayX()

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

Polled delay.

Note

The real delay is always few cycles in excess of the specified value.

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

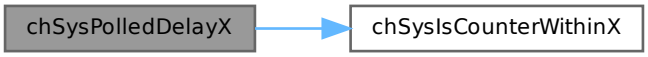
Parameters

in	*cycles*	number of cycles

Function Class:

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

Here is the call graph for this function:



6.3.5.2.3.13 chSysDisable()

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

Raises the system interrupt priority mask to the maximum level.

All the maskable interrupt sources are disabled regardless their hardware priority.

Note

Do not invoke this API from within a kernel lock.

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

Function Class:

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

6.3.5.2.3.14 chSysSuspend()

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

Raises the system interrupt priority mask to system level.

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

Note

Do not invoke this API from within a kernel lock.

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

Function Class:

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

6.3.5.2.3.15 chSysEnable()

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

Lowers the system interrupt priority mask to user level.

All the interrupt sources are enabled.

Note

Do not invoke this API from within a kernel lock.

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

Function Class:

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

6.3.5.2.3.16 chSysLock()

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

Enters the kernel lock state.

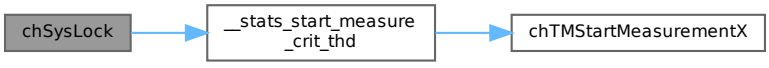
Note

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

Function Class:

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

Here is the call graph for this function:



6.3.5.2.3.17 chSysUnlock()

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

Leaves the kernel lock state.

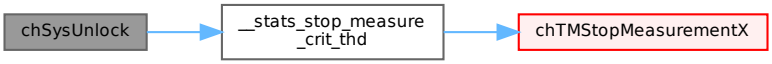
Note

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

Function Class:

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

Here is the call graph for this function:



6.3.5.2.3.18 chSysLockFromISR()

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

Enters the kernel lock state from within an interrupt handler.

Note

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

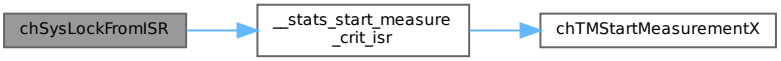
The exact behavior of this function is port-dependent and could not be limited to disabling interrupts.

This API must be invoked exclusively from interrupt handlers.

Function Class:

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

Here is the call graph for this function:



6.3.5.2.3.19 chSysUnlockFromISR()

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

Leaves the kernel lock state from within an interrupt handler.

Note

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

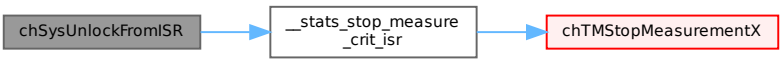
The exact behavior of this function is port-dependent and could not be limited to enabling interrupts.

This API must be invoked exclusively from interrupt handlers.

Function Class:

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

Here is the call graph for this function:



6.3.5.2.3.20 chSysUnconditionalLock()

```
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".

This function is only available if the underlying port supports `port_get_lock_status()` and `port_is_locked()`.

Function Class:

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

Here is the call graph for this function:

6.3.5.2.3.21 chSysUnconditionalUnlock()

```
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".

This function is only available if the underlying port supports `port_get_lock_status()` and `port_is_locked()`.

Function Class:

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

Here is the call graph for this function:

6.3.5.2.3.22 chSysNotifyInstance()

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

Notifies an OS instance to check for reschedule.

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

Parameters

in	*oip*	pointer to the instance to be notified

6.3.5.2.4 Variable Documentation

6.3.5.2.4.1 ch_system

```
ch_system_t ch_system
```

System root object.

6.3.5.2.4.2 ch0

```
CH_SYS_CORE0_MEMORY os_instance_t ch0
```

Core 0 OS instance.

6.3.5.2.4.3 __main_thread_stack_base__

```
stkalign_t __main_thread_stack_base__  [extern]
```

6.3.5.2.4.4 __main_thread_stack_end__

```
stkalign_t __main_thread_stack_end__
```

6.3.5.2.4.5 ch_core0_cfg

```
const os_instance_config_t ch_core0_cfg
```

Initial value:

```
= {
  .name             = "c0",

  .mainthread_base  = &__main_thread_stack_base__,
  .mainthread_end   = &__main_thread_stack_end__,




  .idlethread_base  = THD_WORKING_AREA_BASE(ch_c0_idle_thread_wa),
  .idlethread_end   = THD_WORKING_AREA_END(ch_c0_idle_thread_wa)

}
```

Core 0 OS instance configuration.

6.3.5.2.4.6 ch1

```
CH_SYS_CORE1_MEMORY os_instance_t ch1
```

Core 1 OS instance.

6.3.5.2.4.7 __c1_main_thread_stack_base__

```
stkalign_t __c1_main_thread_stack_base__  [extern]
```

6.3.5.2.4.8 __c1_main_thread_stack_end__

```
stkalign_t __c1_main_thread_stack_end__
```

6.3.5.2.4.9 ch_core1_cfg

```
const os_instance_config_t ch_core1_cfg
```

Initial value:

```
= {
  .name             = "c1",

  .mainthread_base  = &__c1_main_thread_stack_base__,
  .mainthread_end   = &__c1_main_thread_stack_end__,




  .idlethread_base  = THD_WORKING_AREA_BASE(ch_c1_idle_thread_wa),
  .idlethread_end   = THD_WORKING_AREA_END(ch_c1_idle_thread_wa)

}
```

Core 1 OS instance configuration.

6.3.5.3 Memory Alignment

6.3.5.3.1 Detailed Description

Memory Alignment services.

Memory alignment support macros

- #define MEM_ALIGN_MASK(a)

  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)

  Returns whatever a pointer or memory size is aligned.
- #define MEM_IS_VALID_ALIGNMENT(a)

  Returns whatever a constant is a valid alignment.

6.3.5.3.2 Macro Definition Documentation

6.3.5.3.2.1 MEM_ALIGN_MASK

```
#define MEM_ALIGN_MASK(
                a)
```

Value:

```
((size_t)(a) - 1U)
```

Alignment mask constant.

Parameters

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

6.3.5.3.2.2 MEM_ALIGN_PREV

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

Value:

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

Aligns to the previous aligned memory address.

Parameters

in	*p*	variable to be aligned
in	*a*	alignment, must be a power of two

6.3.5.3.2.3 MEM_ALIGN_NEXT

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

Value:

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

Aligns to the next aligned memory address.

Parameters

in	*p*	variable to be aligned
in	*a*	alignment, must be a power of two

6.3.5.3.2.4 MEM_IS_ALIGNED

```
#define MEM_IS_ALIGNED(
              p,
              a)
```

Value:

```
(((size_t)(p) & MEM_ALIGN_MASK(a)) == 0U)
```

Returns whatever a pointer or memory size is aligned.

Parameters

in	*p*	variable to be aligned
in	*a*	alignment, must be a power of two

6.3.5.3.2.5 MEM_IS_VALID_ALIGNMENT

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

Value:

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

Returns whatever a constant is a valid alignment.

Valid alignments are powers of two.

Parameters

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

6.3.5.4 Time and Intervals

6.3.5.4.1 Detailed Description

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

Secure time conversion utilities

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

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)

  Seconds to time interval.
- #define TIME_MS2I(msecs)

  Milliseconds to time interval.
- #define TIME_US2I(usecs)

  Microseconds to time interval.
- #define TIME_I2S(interval)

  Time interval to seconds.
- #define TIME_I2MS(interval)

  Time interval to milliseconds.
- #define TIME_I2US(interval)

  Time interval to microseconds.

Typedefs

- typedef uint64_t systime_t

  Type of system time.
- typedef uint64_t sysinterval_t

  Type of time interval.
- typedef uint64_t systimestamp_t

  Type of a time stamp.
- typedef uint32_t time_secs_t

  Type of seconds.
- typedef uint32_t time_msecs_t

  Type of milliseconds.
- typedef uint32_t time_usecs_t

  Type of microseconds.
- typedef uint64_t time_conv_t

  Type of time conversion variable.

6.3.5.4.2 Macro Definition Documentation

6.3.5.4.2.1 TIME_IMMEDIATE

```
#define TIME_IMMEDIATE ((sysinterval_t)0)
```

Zero interval specification for some functions with a timeout specification.

Note

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

6.3.5.4.2.2 TIME_INFINITE

```
#define TIME_INFINITE ((sysinterval_t)-1)
```

Infinite interval specification for all functions with a timeout specification.

Note

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

6.3.5.4.2.3 TIME_MAX_INTERVAL

```
#define TIME_MAX_INTERVAL ((sysinterval_t)-2)
```

Maximum interval constant usable as timeout.

6.3.5.4.2.4 TIME_MAX_SYSTIME

```
#define TIME_MAX_SYSTIME ((systime_t)-1)
```

Maximum system of system time before it wraps.

6.3.5.4.2.5 TIME_S2I

```
#define TIME_S2I(
              secs)
```

Value:

```
  ((sysinterval_t)((time_conv_t)(secs) * (time_conv_t)CH_CFG_ST_FREQUENCY))
```

Seconds to time interval.

Converts from seconds to system ticks number.

Note

The result is rounded upward to the next tick boundary.

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

Parameters

in	*secs*	number of seconds

Returns

The number of ticks.

Function Class:

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

6.3.5.4.2.6 TIME_MS2I

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

Value:

```
  ((sysinterval_t)((((time_conv_t)(msecs) *                                \
                     (time_conv_t)CH_CFG_ST_FREQUENCY) +                   \
                    (time_conv_t)999) / (time_conv_t)1000))
```

Milliseconds to time interval.

Converts from milliseconds to system ticks number.

Note

The result is rounded upward to the next tick boundary.

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

Parameters

in	*msecs*	number of milliseconds

Returns

The number of ticks.

Function Class:

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

6.3.5.4.2.7 TIME_US2I

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

Value:

```
  ((sysinterval_t)((((time_conv_t)(usecs) *                                \
                     (time_conv_t)CH_CFG_ST_FREQUENCY) +                   \
                    (time_conv_t)999999) / (time_conv_t)1000000))
```

Microseconds to time interval.

Converts from microseconds to system ticks number.

Note

The result is rounded upward to the next tick boundary.

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

Parameters

in	*usecs*	number of microseconds

Returns

The number of ticks.

Function Class:

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

6.3.5.4.2.8 TIME_I2S

```
#define TIME_I2S(
                interval)
```

Value:

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

Time interval to seconds.

Converts from system ticks number to seconds.

Note

The result is rounded up to the next second boundary.

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

Parameters

in	*interval*	interval in ticks

Returns

The number of seconds.

Function Class:

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

6.3.5.4.2.9 TIME_I2MS

```
#define TIME_I2MS(
                interval)
```

Value:

```
  (time_msecs_t)((((time_conv_t)(interval) * (time_conv_t)1000) +           \
                  (time_conv_t)CH_CFG_ST_FREQUENCY - (time_conv_t)1) /      \
                 (time_conv_t)CH_CFG_ST_FREQUENCY)
```

Time interval to milliseconds.

Converts from system ticks number to milliseconds.

Note

The result is rounded up to the next millisecond boundary.

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

Parameters

in	*interval*	interval in ticks

Returns

The number of milliseconds.

Function Class:

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

6.3.5.4.2.10 TIME_I2US

```
#define TIME_I2US(
              interval)
```

Value:

```
  (time_usecs_t)((((time_conv_t)(interval) * (time_conv_t)1000000) +        \
                  (time_conv_t)CH_CFG_ST_FREQUENCY - (time_conv_t)1) /       \
                 (time_conv_t)CH_CFG_ST_FREQUENCY)
```

Time interval to microseconds.

Converts from system ticks number to microseconds.

Note

The result is rounded up to the next microsecond boundary.

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

Parameters

in	*interval*	interval in ticks

Returns

The number of microseconds.

Function Class:

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

6.3.5.4.3 Typedef Documentation

6.3.5.4.3.1 systime_t

```
typedef uint64_t systime_t
```

Type of system time.

Note

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

6.3.5.4.3.2 sysinterval_t

```
typedef uint64_t sysinterval_t
```

Type of time interval.

Note

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

6.3.5.4.3.3 systimestamp_t

```
typedef uint64_t systimestamp_t
```

Type of a time stamp.

6.3.5.4.3.4 time_secs_t

```
typedef uint32_t time_secs_t
```

Type of seconds.

Note

It is selectable in configuration between 16 or 32 bits.

6.3.5.4.3.5 time_msecs_t

```
typedef uint32_t time_msecs_t
```

Type of milliseconds.

Note

It is selectable in configuration between 16 or 32 bits.

6.3.5.4.3.6 time_usecs_t

```
typedef uint32_t time_usecs_t
```

Type of microseconds.

Note

It is selectable in configuration between 16 or 32 bits.

6.3.5.4.3.7 time_conv_t

```
typedef uint64_t time_conv_t
```

Type of time conversion variable.

Note

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

6.3.5.4.4 Function Documentation

6.3.5.4.4.1 chTimeS2I()

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

Seconds to time interval.

Converts from seconds to system ticks number.

Note

The result is rounded upward to the next tick boundary.

Parameters

in	*secs*	number of seconds

Returns

The number of ticks.

Function Class:

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

6.3.5.4.4.2 chTimeMS2I()

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

Milliseconds to time interval.

Converts from milliseconds to system ticks number.

Note

The result is rounded upward to the next tick boundary.

Parameters

in	*msec*	number of milliseconds

Returns

The number of ticks.

Function Class:

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

6.3.5.4.4.3 chTimeUS2I()

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

Microseconds to time interval.

Converts from microseconds to system ticks number.

Note

The result is rounded upward to the next tick boundary.

Parameters

in	*usec*	number of microseconds

Returns

The number of ticks.

Function Class:

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

6.3.5.4.4.4 chTimeI2S()

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

Time interval to seconds.

Converts from system interval to seconds.

Note

The result is rounded up to the next second boundary.

Parameters

in	*interval*	interval in ticks

Returns

The number of seconds.

Function Class:

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

6.3.5.4.4.5 chTimeI2MS()

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

Time interval to milliseconds.

Converts from system interval to milliseconds.

Note

The result is rounded up to the next millisecond boundary.

Parameters

in	*interval*	interval in ticks

Returns

The number of milliseconds.

Function Class:

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

6.3.5.4.4.6 chTimeI2US()

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

Time interval to microseconds.

Converts from system interval to microseconds.

Note

The result is rounded up to the next microsecond boundary.

Parameters

in	*interval*	interval in ticks

Returns

The number of microseconds.

Function Class:

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

6.3.5.4.4.7 chTimeAddX()

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

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

Parameters

in	*systime*	base system time
in	*interval*	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.

6.3.5.4.4.8 chTimeDiffX()

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

Subtracts two system times returning an interval.

Parameters

in	*start*	first system time
in	*end*	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.

6.3.5.4.4.9 chTimeIsInRangeX()

```
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

in	*time*	the time to be verified
in	*start*	the start of the time window (inclusive)
in	*end*	the end of the time window (non inclusive)

Return values

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

Function Class:

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

6.3.5.4.4.10 chTimeStampAddX()

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

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

Parameters

in	*stamp*	base time stamp
in	*interval*	interval to be added

Returns

The new time stamp.

Function Class:

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

6.3.5.4.4.11 chTimeStampDiffX()

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

Subtracts two time stamps returning an interval.

Note

Intervals can then be used for converting in absolute time.

Parameters

in	*start*	first time stamp
in	*end*	second time stamp

Returns

The interval representing the time stamps difference.

Function Class:

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

6.3.5.4.4.12 chTimeStampIsInRangeX()

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

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

Note

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

Parameters

in	*stamp*	the time stamp to be verified
in	*start*	the start of the time stamp window (inclusive)
in	*end*	the end of the time stamp window (non inclusive)

Return values

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

Function Class:

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

6.3.5.5 Virtual Timers

6.3.5.5.1 Detailed Description

Time and Virtual Timers related APIs and services.

Functions

- static void vt_set_alarm (systime_t now, sysinterval_t delay)
  Alarm time setup.
- static void vt_insert_first (virtual_timers_list_t *vtlp, virtual_timer_t *vtp, systime_t now, sysinterval_t delay)
  Inserts a timer as first element in a delta list.
- static void vt_enqueue (virtual_timers_list_t *vtlp, virtual_timer_t *vtp, sysinterval_t delay)
  Enqueues a virtual timer in a virtual timers list.
- void chVTDoSetI (virtual_timer_t *vtp, sysinterval_t delay, vtfunc_t vtfunc, void *par)
  Enables a one-shot virtual timer.
- void chVTDoSetContinuousI (virtual_timer_t *vtp, sysinterval_t delay, vtfunc_t vtfunc, void *par)
  Enables a continuous virtual timer.
- void chVTDoResetI (virtual_timer_t *vtp)
  Disables a Virtual Timer.
- sysinterval_t chVTGetRemainingIntervalI (virtual_timer_t *vtp)
  Returns the remaining time interval before next timer trigger.
- void chVTDoTickI (void)
  Virtual timers ticker.
- systimestamp_t chVTGetTimeStampI (void)
  Generates a monotonic time stamp.
- void chVTResetTimeStampI (void)
  Resets and re-synchronizes the time stamps monotonic counter.
- static void chVTObjectInit (virtual_timer_t *vtp)
  Initializes a `virtual_timer_t` object.
- static systime_t chVTGetSystemTimeX (void)
  Current system time.
- static systime_t chVTGetSystemTime (void)
  Current system time.
- static sysinterval_t chVTTimeElapsedSinceX (systime_t start)
  Returns the elapsed time since the specified start time.
- static bool chVTIsSystemTimeWithinX (systime_t start, systime_t end)
  Checks if the current system time is within the specified time window.
- static bool chVTIsSystemTimeWithin (systime_t start, systime_t end)
  Checks if the current system time is within the specified time window.
- static bool chVTGetTimersStateI (sysinterval_t *timep)
  Returns the time interval until the next timer event.
- static bool chVTIsArmedI (const virtual_timer_t *vtp)
  Returns `true` if the specified timer is armed.
- static bool chVTIsArmed (const virtual_timer_t *vtp)
  Returns `true` if the specified timer is armed.
- static void chVTResetI (virtual_timer_t *vtp)
  Disables a Virtual Timer.
- static void chVTReset (virtual_timer_t *vtp)
  Disables a Virtual Timer.

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

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

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

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

  Enables a continuous virtual timer.
- static sysinterval_t chVTGetReloadIntervalX (virtual_timer_t *vtp)

  Returns the current reload value.
- static void chVTSetReloadIntervalX (virtual_timer_t *vtp, sysinterval_t reload)

  Changes a timer reload time interval.
- static systimestamp_t chVTGetTimeStamp (void)

  Generates a monotonic time stamp.
- static void chVTResetTimeStamp (void)

  Resets and re-synchronizes the time stamps monotonic counter.
- static void __vt_object_init (virtual_timers_list_t *vtlp)

  Virtual Timers instance initialization.

6.3.5.5.2 Function Documentation

6.3.5.5.2.1 vt_set_alarm()

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

Alarm time setup.

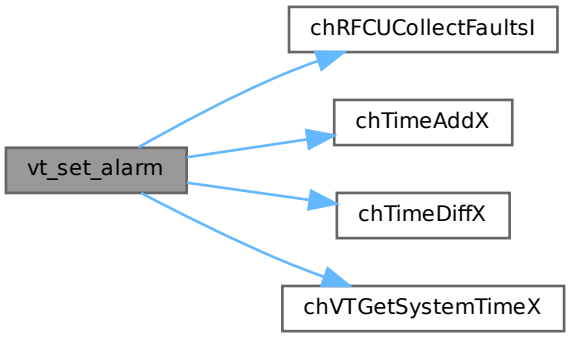
Note

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

Parameters

in	*now*	last known system time
in	*delay*	delay over `now`

Here is the call graph for this function:



6.3.5.5.2.2 vt_insert_first()

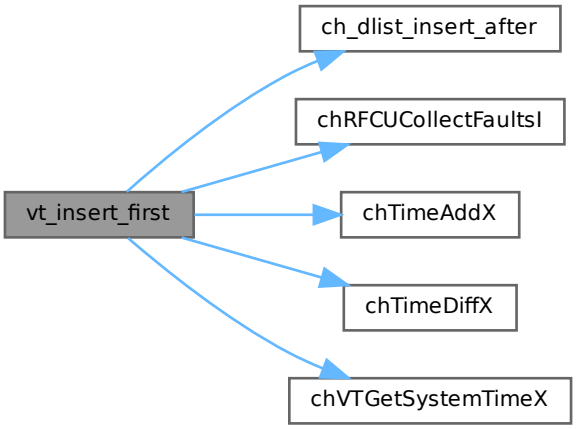
```
void vt_insert_first (
             virtual_timers_list_t * vtlp,
             virtual_timer_t * vtp,
             systime_t now,
             sysinterval_t delay)  [static]
```

Inserts a timer as first element in a delta list.

Note

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

Here is the call graph for this function:

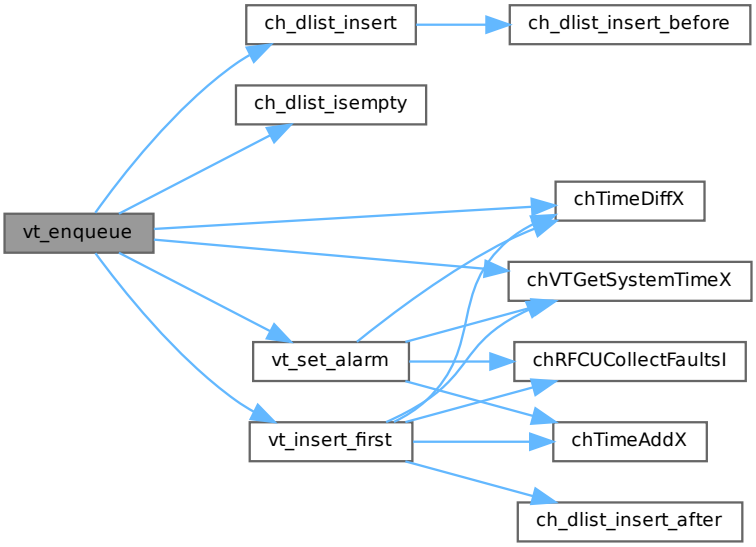


6.3.5.5.2.3 vt_enqueue()

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

Enqueues a virtual timer in a virtual timers list.

Here is the call graph for this function:



6.3.5.5.2.4 chVTDoSetI()

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

Enables a one-shot virtual timer.

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

Precondition

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

Note

The callback function is invoked from interrupt context.

Parameters

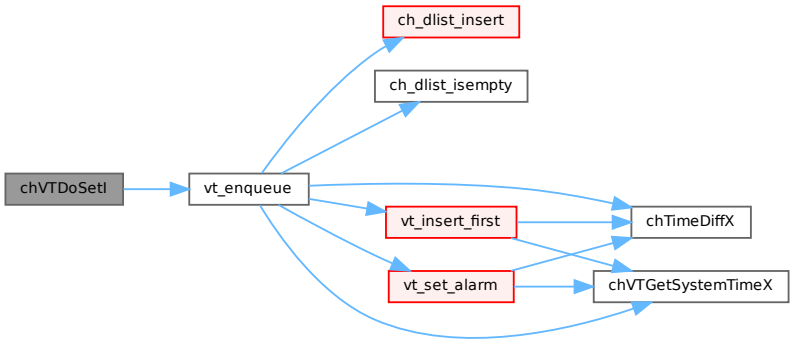
out	*vtp*	pointer to a virtual_timer_t structure
in	*delay*	the number of ticks before the operation timeouts, the special values are handled as follow: • *TIME_INFINITE* is allowed but interpreted as a normal time specification. • *TIME_IMMEDIATE* this value is not allowed.

in	*vtfunc*	the timer callback function. After invoking the callback the timer is disabled and the structure can be disposed or reused.
in	*par*	a parameter that will be passed to the callback function

Function Class:

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

Here is the call graph for this function:



6.3.5.5.2.5 chVTDoSetContinuousI()

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

Enables a continuous virtual timer.

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

Precondition

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

Note

The callback function is invoked from interrupt context.

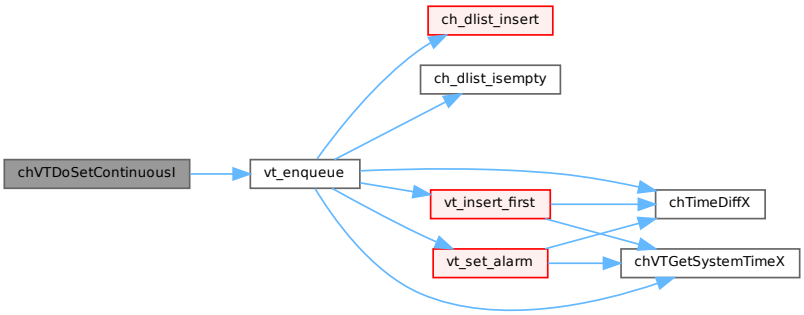
Parameters

out	*vtp*	pointer to a virtual_timer_t structure
in	*delay*	the number of ticks before the operation timeouts, the special values are handled as follow: • *TIME_INFINITE* is allowed but interpreted as a normal time specification. • *TIME_IMMEDIATE* this value is not allowed.
in	*vtfunc*	the timer callback function. After invoking the callback the timer is restarted.
in	*par*	a parameter that will be passed to the callback function

Function Class:

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

Here is the call graph for this function:



6.3.5.5.2.6 chVTDoResetI()

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

Disables a Virtual Timer.

Precondition

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

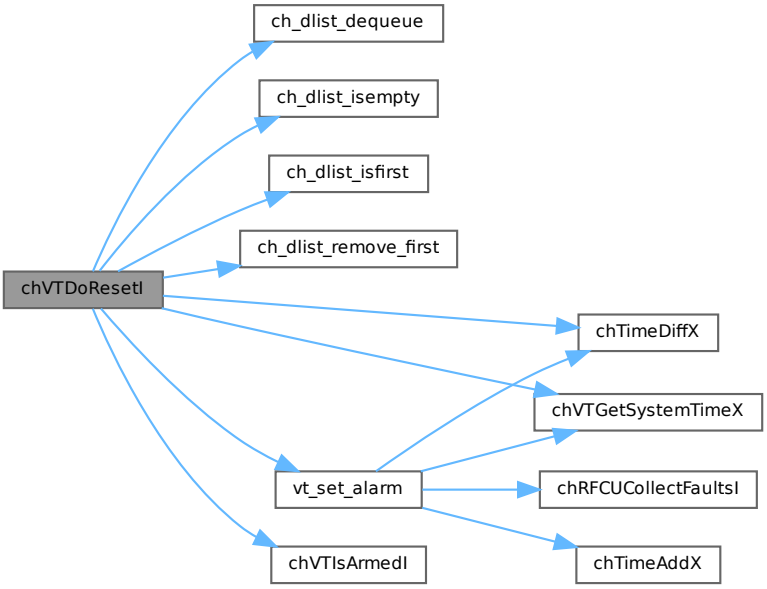
Parameters

in	*vtp*	pointer to a virtual_timer_t structure

Function Class:

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

Here is the call graph for this function:



6.3.5.5.2.7 chVTGetRemainingIntervalI()

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

Returns the remaining time interval before next timer trigger.

Note

This function can be called while the timer is active.

Parameters

in	*vtp*	pointer to a `virtual_timer_t` structure

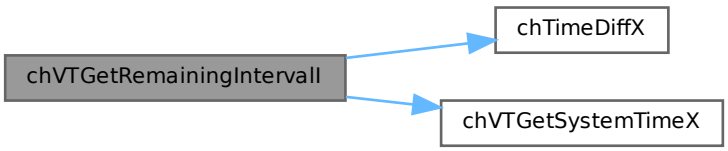
Returns

The remaining time interval.

Function Class:

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

Here is the call graph for this function:



6.3.5.5.2.8 chVTDoTickI()

```
void chVTDoTickI (
             void )
```

Virtual timers ticker.

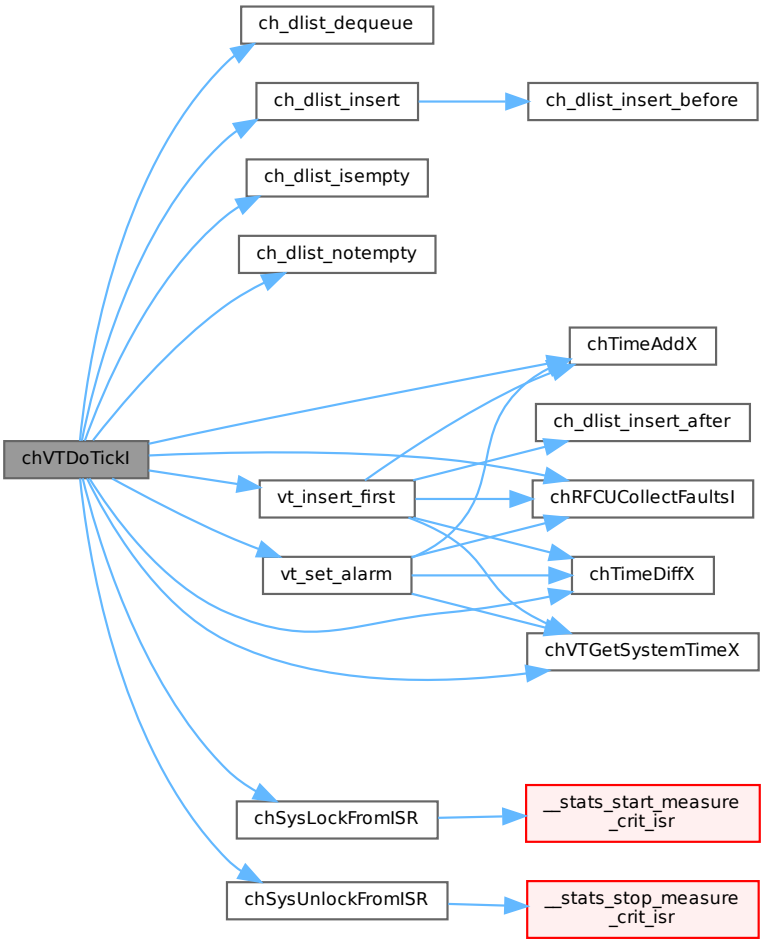
Note

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

Function Class:

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

Here is the call graph for this function:



6.3.5.5.2.9 chVTGetTimeStampI()

```
systimestamp_t chVTGetTimeStampI (
            void )
```

Generates a monotonic time stamp.

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

Note

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

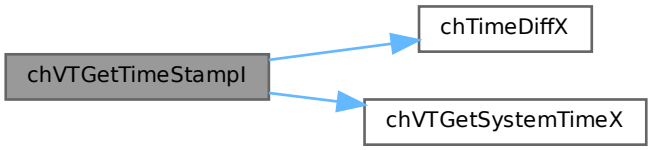
Returns

The time stamp.

Function Class:

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

Here is the call graph for this function:



6.3.5.5.2.10 chVTResetTimeStampI()

```
void chVTResetTimeStampI (
            void )
```

Resets and re-synchronizes the time stamps monotonic counter.

Function Class:

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

Here is the call graph for this function:

6.3.5.5.2.11 chVTObjectInit()

```
void chVTObjectInit (
            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:

Object or module nitializer function.

6.3.5.5.2.12 chVTGetSystemTimeX()

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

Current system time.

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

Note

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

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

Returns

The system time in ticks.

Function Class:

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

6.3.5.5.2.13 chVTGetSystemTime()

```
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.

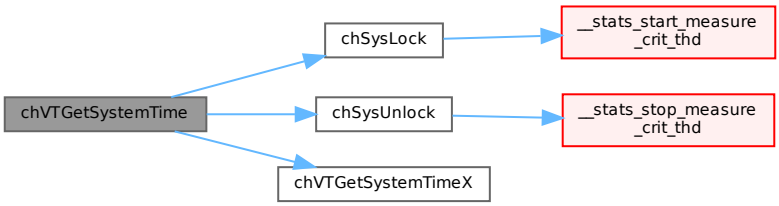
Returns

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:



6.3.5.5.2.14 chVTTimeElapsedSinceX()

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

Returns the elapsed time since the specified start time.

Parameters

in	*start*	start time

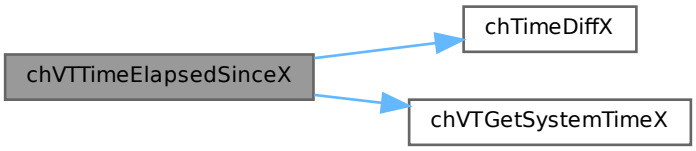
Returns

The elapsed time.

Function Class:

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

Here is the call graph for this function:



6.3.5.5.2.15 chVTIsSystemTimeWithinX()

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

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

Note

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

Parameters

in	*start*	the start of the time window (inclusive)
in	*end*	the end of the time window (non inclusive)

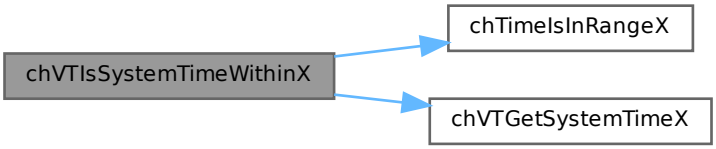
Return values

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

Function Class:

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

Here is the call graph for this function:



6.3.5.5.2.16 chVTIsSystemTimeWithin()

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

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

Note

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

Parameters

in	*start*	the start of the time window (inclusive)
in	*end*	the end of the time window (non inclusive)

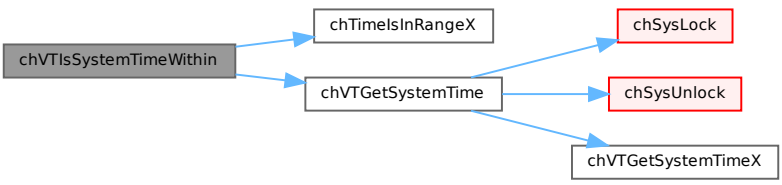
Return values

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

Function Class:

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

Here is the call graph for this function:



6.3.5.5.2.17 chVTGetTimersStateI()

```
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.

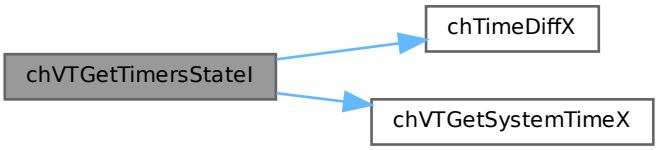
Return values

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:



6.3.5.5.2.18 chVTIsArmedI()

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

Returns `true` if the specified timer is armed.

Precondition

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

Parameters

in	*vtp*	the virtual_timer_t structure pointer

Returns

true if the timer is armed.

Function Class:

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

6.3.5.5.2.19 chVTIsArmed()

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

Returns `true` if the specified timer is armed.

Precondition

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

Parameters

in	*vtp*	the virtual_timer_t structure pointer

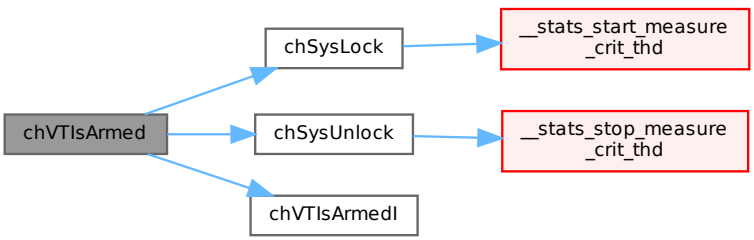
Returns

true if the timer is armed.

Function Class:

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

Here is the call graph for this function:



6.3.5.5.2.20 chVTResetI()

```
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()`.

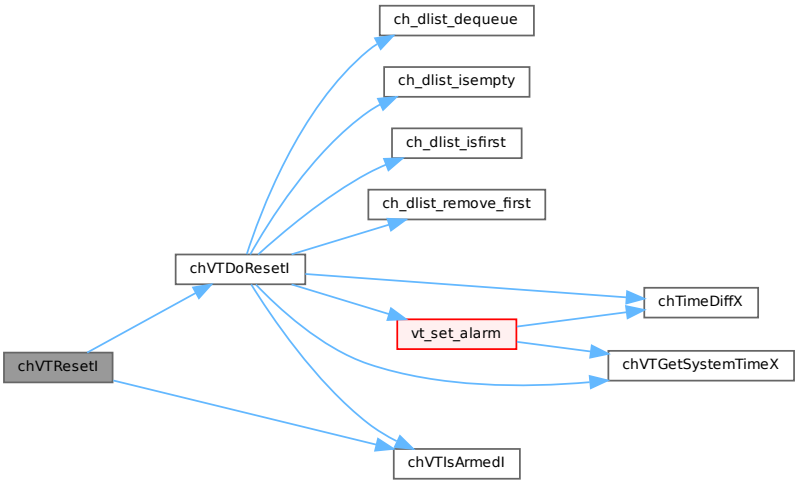
Parameters

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:



6.3.5.5.2.21 chVTReset()

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

Disables a Virtual Timer.

Note

The timer is first checked and disabled only if armed.

Precondition

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

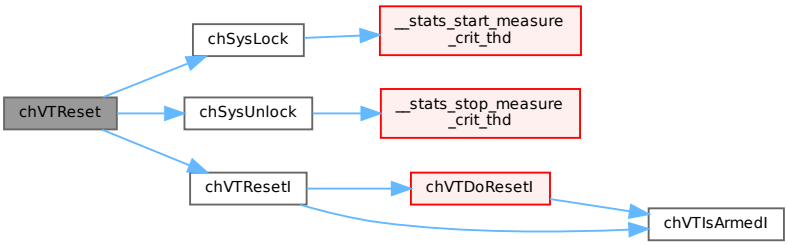
Parameters

in	*vtp*	the virtual_timer_t structure pointer

Function Class:

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

Here is the call graph for this function:



6.3.5.5.2.22 chVTSetI()

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

Enables a one-shot virtual timer.

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

Precondition

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

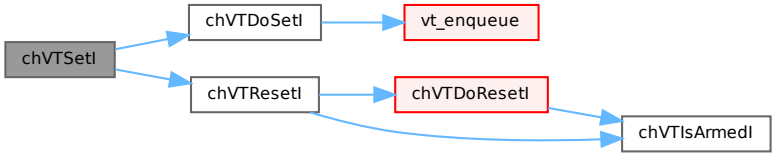
Parameters

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

Function Class:

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

Here is the call graph for this function:



6.3.5.5.2.23 chVTSet()

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

Enables a one-shot virtual timer.

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

Precondition

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

Parameters

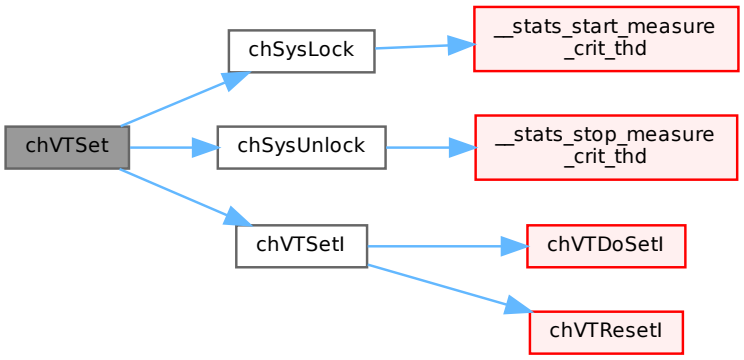
in	*vtp*	the virtual_timer_t structure pointer

in	*delay*	the number of ticks before the operation timeouts, the special values are handled as follow: • *TIME_INFINITE* is allowed but interpreted as a normal time specification. • *TIME_IMMEDIATE* this value is not allowed.
in	*vtfunc*	the timer callback function. After invoking the callback the timer is disabled and the structure can be disposed or reused.
in	*par*	a parameter that will be passed to the callback function

Function Class:

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

Here is the call graph for this function:



6.3.5.5.2.24 chVTSetContinuousI()

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

Enables a continuous virtual timer.

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

Precondition

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

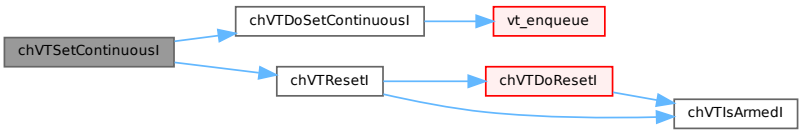
Parameters

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

Function Class:

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

Here is the call graph for this function:



6.3.5.5.2.25 chVTSetContinuous()

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

Enables a continuous virtual timer.

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

Precondition

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

Parameters

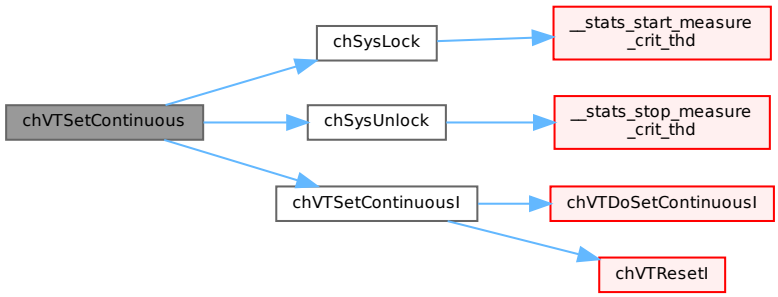
in	*vtp*	the `virtual_timer_t` structure pointer
in	*delay*	the number of ticks before the operation timeouts, the special values are handled as follow: • *TIME_INFINITE* is allowed but interpreted as a normal time specification. • *TIME_IMMEDIATE* this value is not allowed.

in	*vtfunc*	the timer callback function. After invoking the callback the timer is disabled and the structure can be disposed or reused.
in	*par*	a parameter that will be passed to the callback function

Function Class:

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

Here is the call graph for this function:



6.3.5.5.2.26 chVTGetReloadIntervalX()

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

Returns the current reload value.

Parameters

in	*vtp*	the `virtual_timer_t` structure pointer

Returns

The reload value.

Function Class:

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

6.3.5.5.2.27 chVTSetReloadIntervalX()

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

Changes a timer reload time interval.

Note

This function is meant to be called from a timer callback, it does nothing in any other context.

Calling this function from a one-shot timer callback turns it into a continuous timer.

Parameters

in	*vtp*	the virtual_timer_t structure pointer
in	*reload*	the new reload value, zero means no reload

Function Class:

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

6.3.5.5.2.28 chVTGetTimeStamp()

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

Generates a monotonic time stamp.

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

Note

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

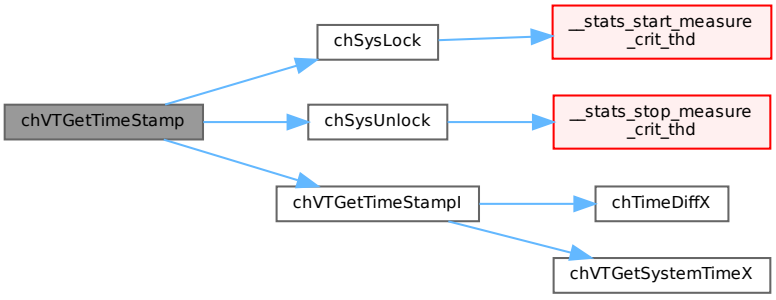
Returns

The time stamp.

Function Class:

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

Here is the call graph for this function:



6.3.5.5.2.29 chVTResetTimeStamp()

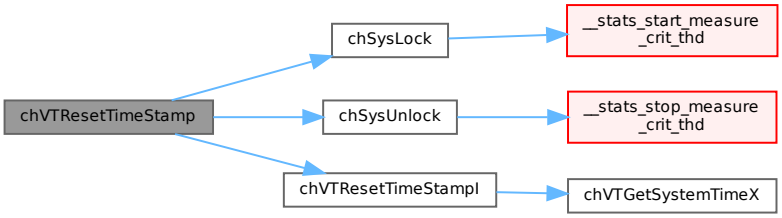
```
void chVTResetTimeStamp (
            void )  [inline], [static]
```

Resets and re-synchronizes the time stamps monotonic counter.

Function Class:

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

Here is the call graph for this function:



6.3.5.5.2.30 __vt_object_init()

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

Virtual Timers instance initialization.

Note

Internal use only.

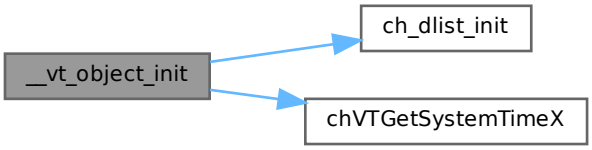
Parameters

out	*vtlp*	pointer to the virtual_timers_list_t structure

Function Class:

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

Here is the call graph for this function:



6.3.5.6 Threads

6.3.5.6.1 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)
  Data part of a static threads queue object initializer.
- #define THREADS_QUEUE_DECL(name)
  Static threads queue object initializer.

Working Areas

- #define THD_WORKING_AREA_SIZE(n)
  Calculates the total Working Area size.
- #define THD_WORKING_AREA(s, n)
  Static working area allocation.
- #define THD_WORKING_AREA_BASE(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)
  Thread declaration macro.

Threads initializers

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

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

  Thread descriptor initializer with no affinity.

Macro Functions

- #define chThdSleepSeconds(sec)

  Delays the invoking thread for the specified number of seconds.
- #define chThdSleepMilliseconds(msec)

  Delays the invoking thread for the specified number of milliseconds.
- #define chThdSleepMicroseconds(usec)

  Delays the invoking thread for the specified number of microseconds.

Data Structures

- struct thread_descriptor_t

  Type of a thread descriptor.

Typedefs

- typedef void(∗ tfunc_t) (void ∗p)

  Thread function.

Functions

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

  Initializes a thread structure.
- void __thd_stackfill (uint8_t ∗startp, uint8_t ∗endp)

  Stack fill utility.
- thread_t ∗ chThdCreateSuspendedI (const thread_descriptor_t ∗tdp)

  Creates a new thread.
- thread_t ∗ chThdCreateSuspended (const thread_descriptor_t ∗tdp)

  Creates a new thread.
- thread_t ∗ chThdCreateI (const thread_descriptor_t ∗tdp)

  Creates a new thread.
- thread_t ∗ chThdCreate (const thread_descriptor_t ∗tdp)

  Creates a new thread.
- thread_t ∗ chThdCreateStatic (void ∗wsp, size_t size, tprio_t prio, tfunc_t pf, void ∗arg)

  Creates a new thread.
- thread_t ∗ chThdStart (thread_t ∗tp)

  Starts a thread created with `chThdCreateSuspended()`.
- 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.

6.3.5.6.2 Macro Definition Documentation

6.3.5.6.2.1 __THREADS_QUEUE_DATA

```
#define __THREADS_QUEUE_DATA(
              name)
```

Value:

```
{__CH_QUEUE_DATA(name)}
```

Data part of a static threads queue object initializer.

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

Parameters

in	*name*	the name of the threads queue variable

6.3.5.6.2.2 THREADS_QUEUE_DECL

```
#define THREADS_QUEUE_DECL(
              name)
```

Value:

```
  threads_queue_t name = __THREADS_QUEUE_DATA(name)
```

Static threads queue object initializer.

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

Parameters

in	*name*	the name of the threads queue variable

6.3.5.6.2.3 THD_WORKING_AREA_SIZE

```
#define THD_WORKING_AREA_SIZE(
              n)
```

Value:

```
  MEM_ALIGN_NEXT(sizeof(thread_t) + PORT_WA_SIZE(n), PORT_STACK_ALIGN)
```

Calculates the total Working Area size.

Parameters

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

Returns

The total used memory in bytes.

Function Class:

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

6.3.5.6.2.4 THD_WORKING_AREA

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

Value:

```
PORT_WORKING_AREA(s, n)
```

Static working area allocation.

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

Parameters

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

Function Class:

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

6.3.5.6.2.5 THD_WORKING_AREA_BASE

```
#define THD_WORKING_AREA_BASE(
              s)
```

Value:

```
((stkalign_t *)(s))
```

Base of a working area casted to the correct type.

Parameters

in	*s*	name of the working area

6.3.5.6.2.6 THD_WORKING_AREA_END

```
#define THD_WORKING_AREA_END(
              s)
```

Value:

```
                                (THD_WORKING_AREA_BASE(s) +                \
                                (sizeof (s) / sizeof (stkalign_t)))
```

End of a working area casted to the correct type.

Parameters

in	*s*	name of the working area

6.3.5.6.2.7 THD_FUNCTION

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

Value:

```
PORT_THD_FUNCTION(tname, arg)
```

Thread declaration macro.

Note

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

6.3.5.6.2.8 THD_DESCRIPTOR

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

Value:

```
  {               \
  (name),                                                 \
  (wbase),                                                \
  (wend),                                                 \
  (prio),                                                 \
  (funcp),                                                \
  (arg),                                                  \
  NULL                                                    \
}
```

Thread descriptor initializer with no affinity.

Parameters

in	*name*	thread name
in	*wbase*	pointer to the working area base
in	*wend*	pointer to the working area end
in	*prio*	thread priority
in	*funcp*	thread function pointer
in	*arg*	thread argument

6.3.5.6.2.9 THD_DESCRIPTOR_AFFINITY

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

Value:

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

Thread descriptor initializer with no affinity.

Parameters

in	*name*	thread name
in	*wbase*	pointer to the working area base
in	*wend*	pointer to the working area end
in	*prio*	thread priority
in	*funcp*	thread function pointer
in	*arg*	thread argument
in	*oip*	instance affinity

6.3.5.6.2.10 chThdSleepSeconds

```
#define chThdSleepSeconds(
              sec)
```

Value:

```
chThdSleep(TIME_S2I(sec))
```

Delays the invoking thread for the specified number of seconds.

Note

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

The maximum specifiable value is implementation dependent.

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

Parameters

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

Function Class:

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

6.3.5.6.2.11 chThdSleepMilliseconds

```
#define chThdSleepMilliseconds(
              msec)
```

Value:

```
chThdSleep(TIME_MS2I(msec))
```

Delays the invoking thread for the specified number of milliseconds.

Note

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

The maximum specifiable value is implementation dependent.

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

Parameters

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

Function Class:

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

6.3.5.6.2.12 chThdSleepMicroseconds

```
#define chThdSleepMicroseconds(
              usec)
```

Value:

```
chThdSleep(TIME_US2I(usec))
```

Delays the invoking thread for the specified number of microseconds.

Note

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

The maximum specifiable value is implementation dependent.

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

Parameters

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

Function Class:

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

6.3.5.6.3 Typedef Documentation

6.3.5.6.3.1 tfunc_t

```
typedef void(* tfunc_t) (void *p)
```

Thread function.

6.3.5.6.4 Function Documentation

6.3.5.6.4.1 __thd_object_init()

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

Initializes a thread structure.

Note

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

Parameters

in	*oip*	pointer to the OS instance
in	*tp*	pointer to the thread
in	*name*	thread name
in	*prio*	the priority level for the new thread

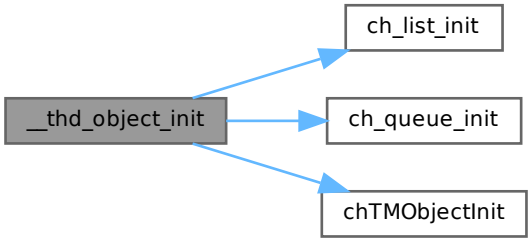
Returns

The same thread pointer passed as parameter.

Function Class:

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

Here is the call graph for this function:



6.3.5.6.4.2 __thd_stackfill()

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

Stack fill utility.

Parameters

in	*startp*	first address to fill
in	*endp*	last address to fill +1

Function Class:

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

6.3.5.6.4.3 chThdCreateSuspendedI()

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

Creates a new thread.

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

Postcondition

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

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

Note

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

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

Parameters

in	*tdp*	pointer to the thread descriptor

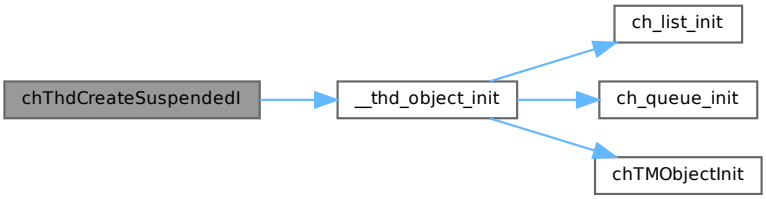
Returns

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

Function Class:

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

Here is the call graph for this function:



6.3.5.6.4.4 chThdCreateSuspended()

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

Creates a new thread.

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

Postcondition

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

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

Note

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

Parameters

in	*tdp*	pointer to the thread descriptor

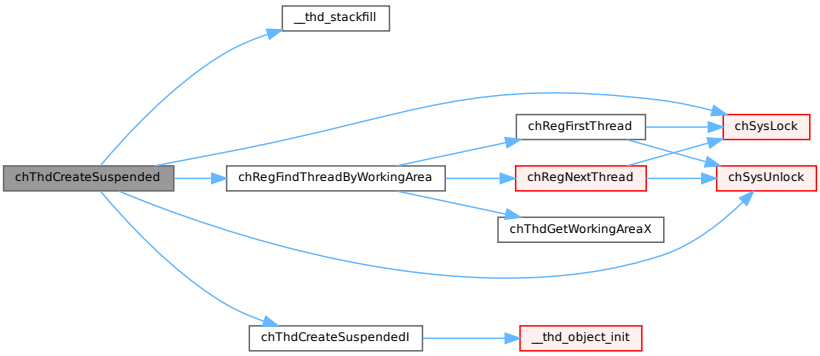
Returns

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

Function Class:

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

Here is the call graph for this function:



6.3.5.6.4.5 chThdCreateI()

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

Creates a new thread.

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.

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

`in`	*tdp*	pointer to the thread descriptor

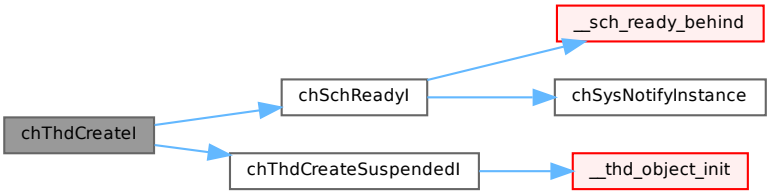
Returns

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

Function Class:

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

Here is the call graph for this function:



6.3.5.6.4.6 chThdCreate()

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

Creates a new thread.

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

in	*tdp*	pointer to the thread descriptor

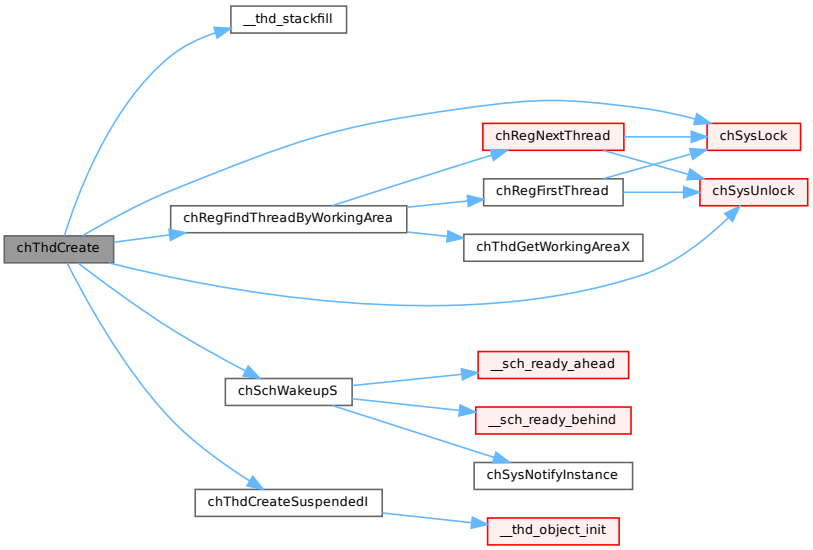
Returns

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

Function Class:

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

Here is the call graph for this function:



6.3.5.6.4.7 chThdCreateStatic()

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

Creates a new thread.

Postcondition

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

Note

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

Parameters

out	*wsp*	pointer to a working area dedicated to the thread stack

in	*size*	size of the working area
in	*prio*	priority level for the new thread
in	*pf*	thread function
in	*arg*	an argument passed to the thread function. It can be NULL.

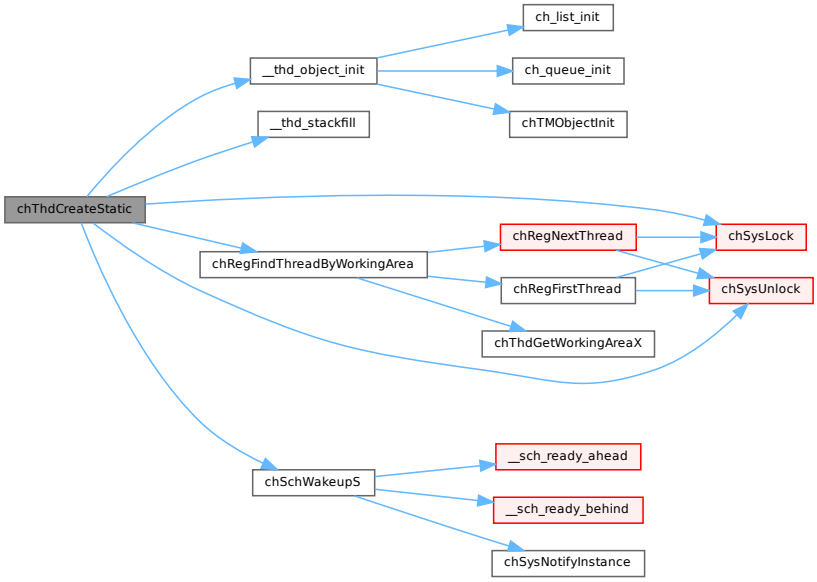
Returns

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

Function Class:

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

Here is the call graph for this function:



6.3.5.6.4.8 chThdStart()

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

Starts a thread created with chThdCreateSuspended().

Parameters

in	*tp*	pointer to the thread

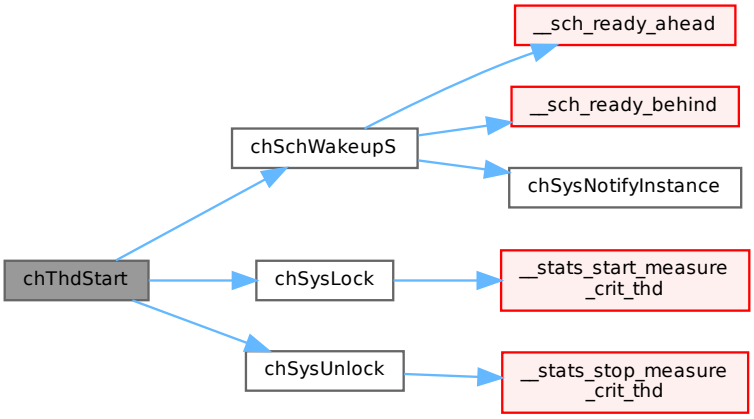
Returns

Thread to be started.

Function Class:

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

Here is the call graph for this function:



6.3.5.6.4.9 chThdAddRef()

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

Adds a reference to a thread object.

Precondition

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

Parameters

`in`	*tp*	pointer to the thread

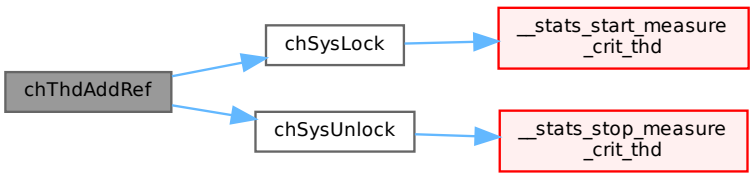
Returns

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

Function Class:

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

Here is the call graph for this function:



6.3.5.6.4.10 chThdRelease()

```
void chThdRelease (
             thread_t * tp)
```

Releases a reference to a thread object.

If the references counter reaches zero **and** the thread is in the `CH_STATE_FINAL` state then the thread's memory is returned to the proper allocator and the thread is removed from the registry.
Threads whose counter reaches zero and are still active become "detached". Detached static threads will be removed from the registry on termination. Detached non-static threads can only be removed by performing a registry scan operation.

Precondition

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

Note

Static threads are not affected, only removed from the registry.

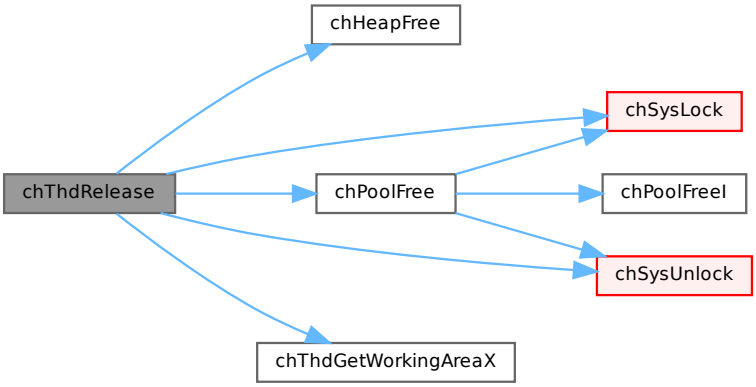
Parameters

`in`	*tp*	pointer to the thread

Function Class:

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

Here is the call graph for this function:



6.3.5.6.4.11 chThdExit()

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

Terminates the current thread.

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

Postcondition

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

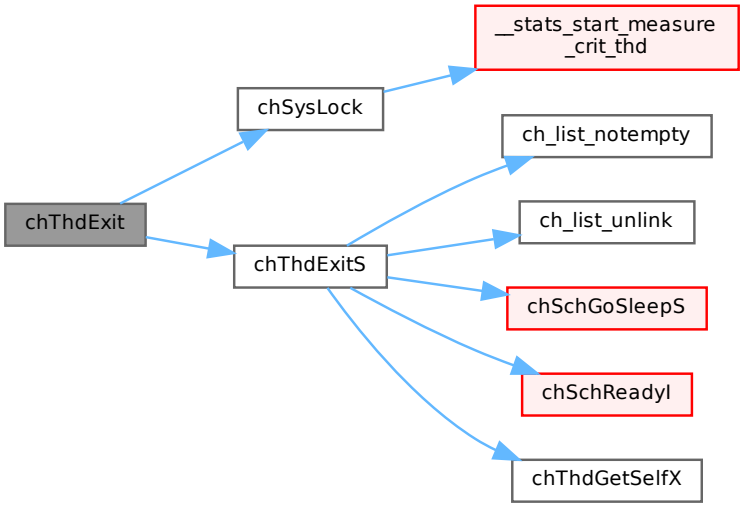
Parameters

`in`	*msg*	thread exit code

Function Class:

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

Here is the call graph for this function:



6.3.5.6.4.12 chThdExitS()

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

Terminates the current thread.

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

Postcondition

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

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

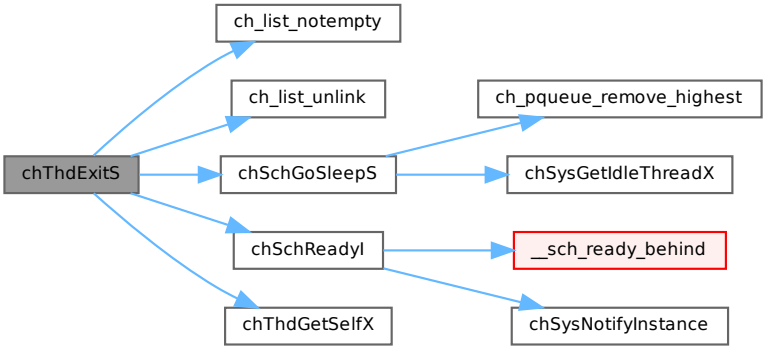
Parameters

`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.

Here is the call graph for this function:



6.3.5.6.4.13 chThdWait()

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

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

This function waits for the specified thread to terminate then decrements its reference counter, if the counter reaches zero then the thread working area is returned to the proper allocator and the thread is removed from the registry.

Precondition

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

Postcondition

Enabling `chThdWait()` requires 2-4 (depending on the architecture) extra bytes in the `thread_t` structure.

Note

If `CH_CFG_USE_DYNAMIC` is not specified this function just waits for the thread termination, no memory allocators are involved.

Parameters

`in`	*tp*	pointer to the thread

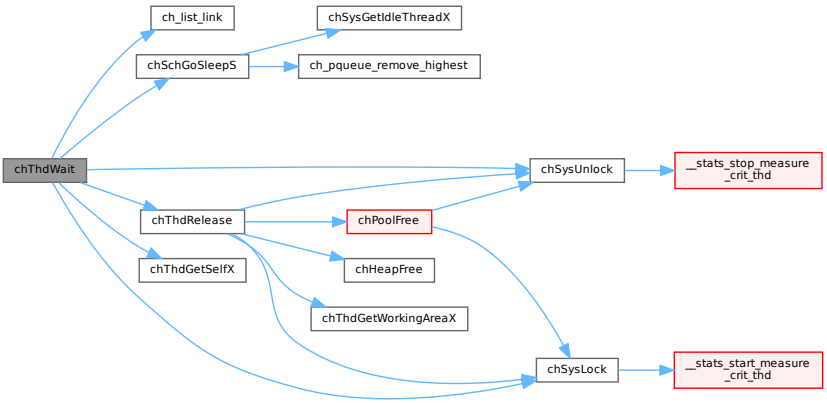
Returns

The exit code from the terminated thread.

Function Class:

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

Here is the call graph for this function:



6.3.5.6.4.14 chThdSetPriority()

```
tprio_t chThdSetPriority (
            tprio_t newprio)
```

Changes the running thread priority level then reschedules if necessary.

Note

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

Parameters

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

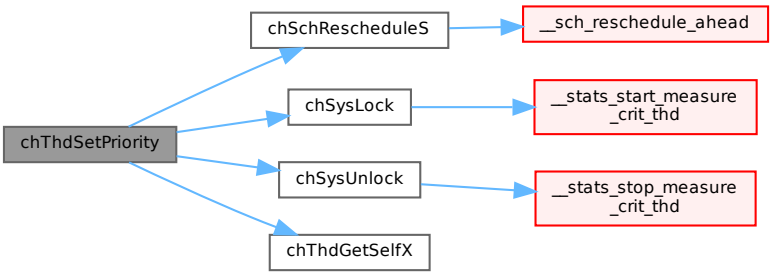
Returns

The old priority level.

Function Class:

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

Here is the call graph for this function:



6.3.5.6.4.15 chThdTerminate()

```
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.

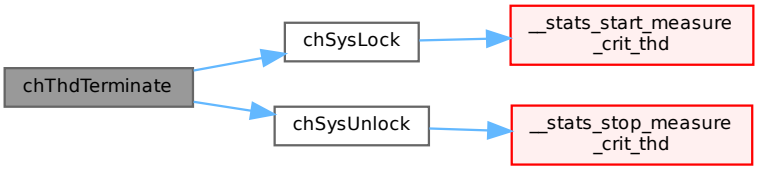
Parameters

`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:



6.3.5.6.4.16 chThdSleep()

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

Suspends the invoking thread for the specified time.

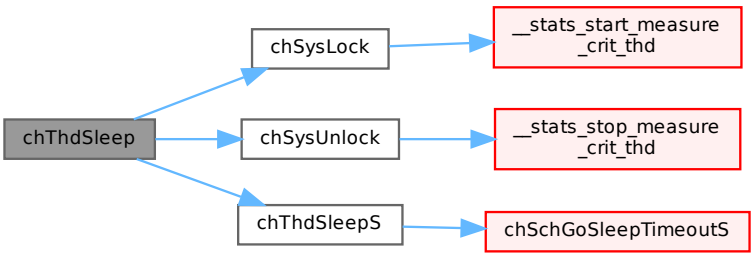
Parameters

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

Function Class:

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

Here is the call graph for this function:



6.3.5.6.4.17 chThdSleepUntil()

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

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

Note

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

See also

chThdSleepUntilWindowed()

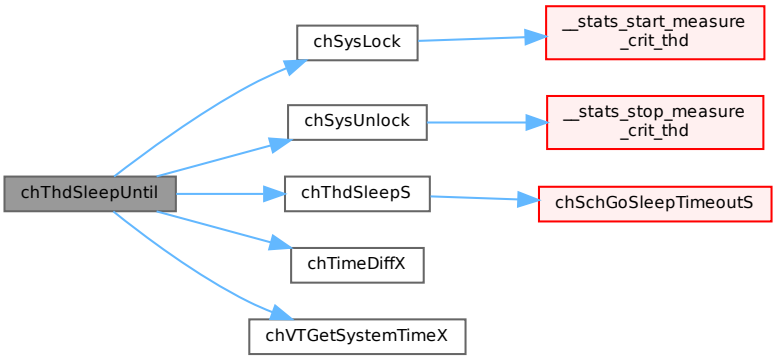
Parameters

in	*time*	absolute system time

Function Class:

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

Here is the call graph for this function:



6.3.5.6.4.18 chThdSleepUntilWindowed()

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

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

Note

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

See also

chThdSleepUntil()

Parameters

in	*prev*	absolute system time of the previous deadline
in	*next*	absolute system time of the next deadline

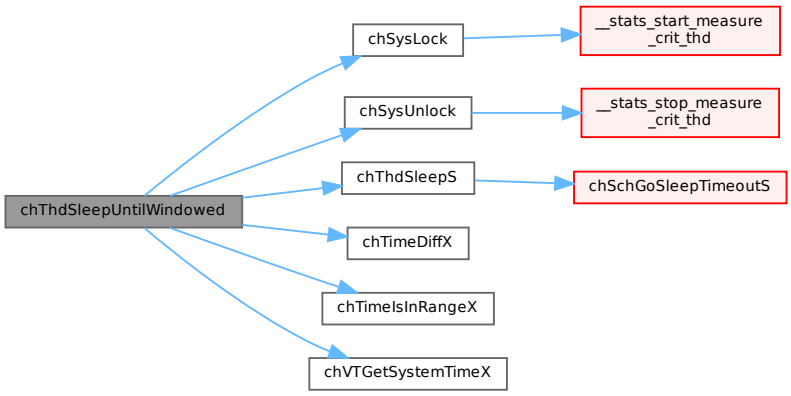
Returns

the `next` parameter

Function Class:

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

Here is the call graph for this function:



6.3.5.6.4.19 chThdYield()

```
void chThdYield (
              void )
```

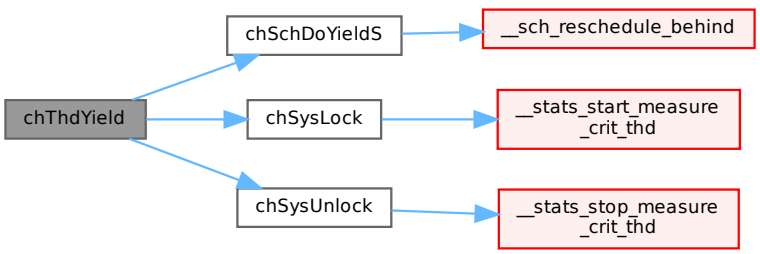
Yields the time slot.

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

Function Class:

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

Here is the call graph for this function:



6.3.5.6.4.20 chThdSuspendS()

```
msg_t chThdSuspendS (
            thread_reference_t * trp)
```

Sends the current thread sleeping and sets a reference variable.

Note

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

Parameters

in	*trp*	a pointer to a thread reference object

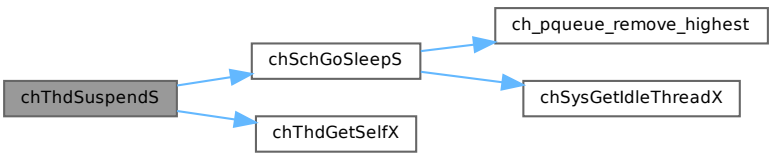
Returns

The wake up message.

Function Class:

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

Here is the call graph for this function:



6.3.5.6.4.21 chThdSuspendTimeoutS()

```
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

in	*trp*	a pointer to a thread reference object
in	*timeout*	the timeout in system ticks, the special values are handled as follow: • *TIME_INFINITE* the thread enters an infinite sleep state. • *TIME_IMMEDIATE* the thread is not suspended and the function returns MSG_TIMEOUT as if a timeout occurred.

Returns

The wake up message.

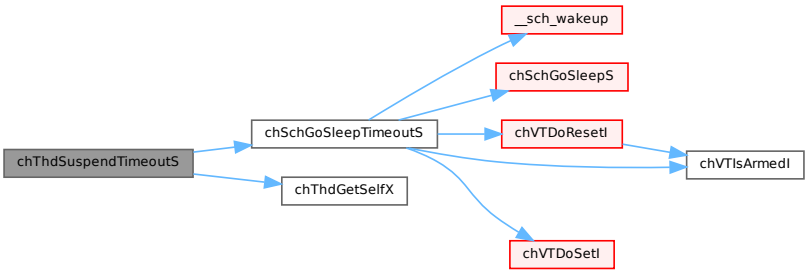
Return values

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:



6.3.5.6.4.22 chThdResumeI()

```
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.

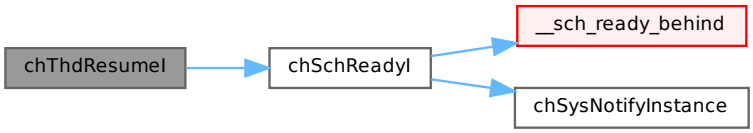
Parameters

in	*trp*	a pointer to a thread reference 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:



6.3.5.6.4.23 chThdResumeS()

```
void chThdResumeS (
             thread_reference_t * trp,
             msg_t msg)
```

Wakes up a thread waiting on a thread reference object.

Note

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

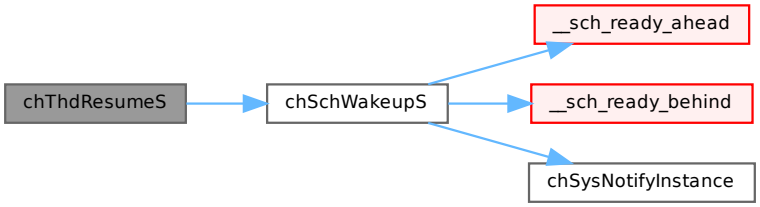
Parameters

in	*trp*	a pointer to a thread reference 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:



6.3.5.6.4.24 chThdResume()

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

Wakes up a thread waiting on a thread reference object.

Note

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

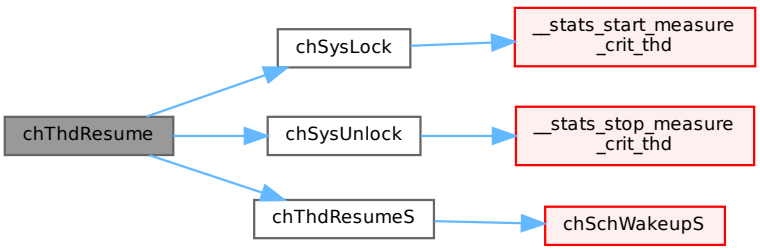
Parameters

in	*trp*	a pointer to a thread reference object
in	*msg*	the message code

Function Class:

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

Here is the call graph for this function:



6.3.5.6.4.25 chThdEnqueueTimeoutS()

```
msg_t chThdEnqueueTimeoutS (
            threads_queue_t * tqp,
            sysinterval_t timeout)
```

Enqueues the caller thread on a threads queue object.

The caller thread is enqueued and put to sleep until it is dequeued or the specified timeouts expires.

Parameters

in	*tqp*	pointer to a `threads_queue_t` object
in	*timeout*	the timeout in system ticks, the special values are handled as follow: • *TIME_INFINITE* the thread enters an infinite sleep state. • *TIME_IMMEDIATE* the thread is not enqueued and the function returns `MSG_TIMEOUT` as if a timeout occurred.

Returns

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

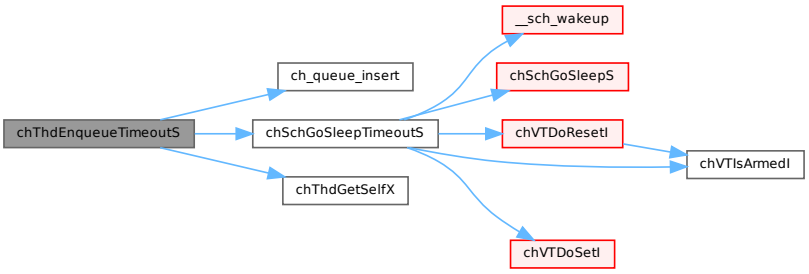
Return values

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:



6.3.5.6.4.26 chThdDequeueNextI()

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

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

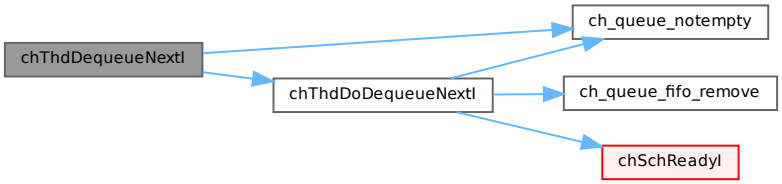
Parameters

in	*tqp*	pointer to a threads_queue_t 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:



6.3.5.6.4.27 chThdDequeueAllI()

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

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

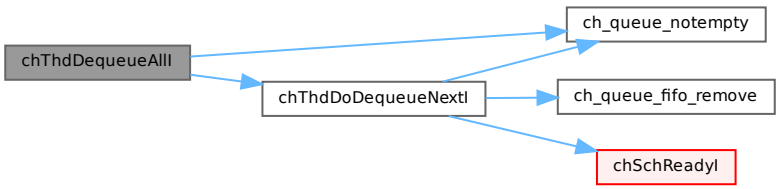
Parameters

in	*tqp*	pointer to a threads_queue_t 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:



6.3.5.6.4.28 chThdGetSelfX()

```
thread_t * chThdGetSelfX (
             void )  [inline], [static]
```

Returns a pointer to the current thread_t.

Returns

A pointer to the current thread.

Function Class:

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

6.3.5.6.4.29 chThdGetPriorityX()

```
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:

6.3.5.6.4.30 chThdGetTicksX()

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

Returns the number of ticks consumed by the specified thread.

Note

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

Parameters

in	*tp*	pointer to the thread

Returns

The number of consumed system ticks.

Function Class:

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

6.3.5.6.4.31 chThdGetWorkingAreaX()

```
stkalign_t * chThdGetWorkingAreaX (
             thread_t * tp)  [inline], [static]
```

Returns the working area base of the specified thread.

Parameters

in	*tp*	pointer to the thread

Returns

The working area base pointer.

Function Class:

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

6.3.5.6.4.32 chThdTerminatedX()

```
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.

6.3.5.6.4.33 chThdShouldTerminateX()

```
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:

6.3.5.6.4.34 chThdStartI()

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

Resumes a thread created with `chThdCreateI()`.

Parameters

in	*tp*	pointer to the thread

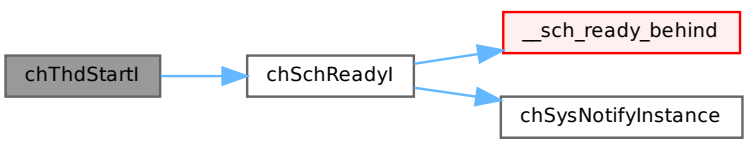
Returns

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

Function Class:

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

Here is the call graph for this function:



6.3.5.6.4.35 chThdSleepS()

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

Suspends the invoking thread for the specified number of ticks.

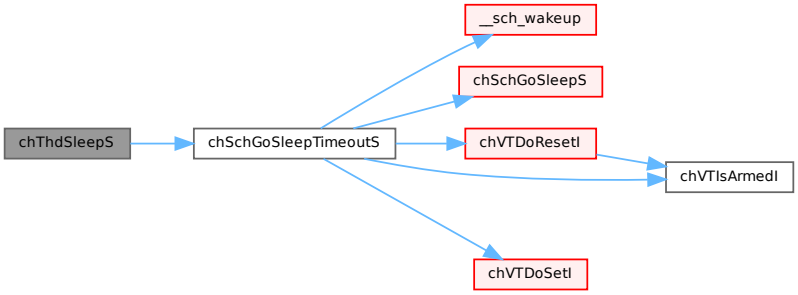
Parameters

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

Function Class:

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

Here is the call graph for this function:



6.3.5.6.4.36 chThdQueueObjectInit()

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

Initializes a threads queue object.

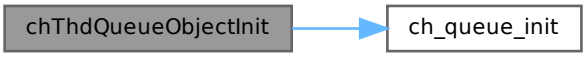
Parameters

out	*tqp*	pointer to the threads queue object

Function Class:

Object or module nitializer function.

Here is the call graph for this function:



6.3.5.6.4.37 chThdQueueIsEmptyI()

```
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:

6.3.5.6.4.38 chThdDoDequeueNextI()

```
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.

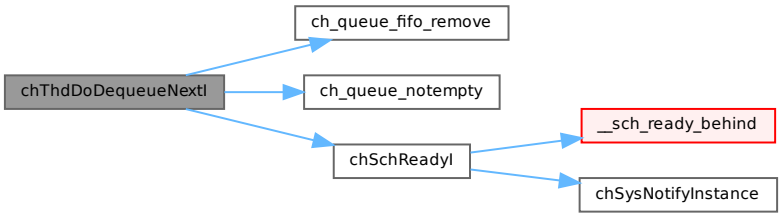
Parameters

in	*tqp*	pointer to the threads queue object
in	*msg*	the message code

Function Class:

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

Here is the call graph for this function:



6.3.5.7 Time Measurement

6.3.5.7.1 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.

Macros

- #define TM_CALIBRATION_LOOP 4U

  Number of iterations in the calibration loop.

Functions

- static void tm_stop (time_measurement_t *tmp, rtcnt_t now, rtcnt_t offset)
- void chTMObjectInit (time_measurement_t *tmp)

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

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

  Stops a measurement.
- NOINLINE void chTMChainMeasurementToX (time_measurement_t *tmp1, time_measurement_t *tmp2)

  Stops a measurement and chains to the next one using the same time stamp.
- static void __tm_calibration_object_init (tm_calibration_t *tcp)

  Time measurement initialization.

6.3.5.7.2 Macro Definition Documentation

6.3.5.7.2.1 TM_CALIBRATION_LOOP

```
#define TM_CALIBRATION_LOOP 4U
```

Number of iterations in the calibration loop.

Note

This is required in order to assess the best result in architectures with instruction cache.

6.3.5.7.3 Function Documentation

6.3.5.7.3.1 tm_stop()

```
void tm_stop (
             time_measurement_t * tmp,
             rtcnt_t now,
             rtcnt_t offset)  [inline], [static]
```

6.3.5.7.3.2 chTMObjectInit()

```
void chTMObjectInit (
             time_measurement_t * tmp)
```

Initializes a `TimeMeasurement` object.

Parameters

out	*tmp*	pointer to a TimeMeasurement structure

Function Class:

Object or module nitializer function.

6.3.5.7.3.3 chTMStartMeasurementX()

```
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.

6.3.5.7.3.4 chTMStopMeasurementX()

```
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.

Here is the call graph for this function:

6.3.5.7.3.5 chTMChainMeasurementToX()

```
NOINLINE void chTMChainMeasurementToX (
              time_measurement_t * tmp1,
              time_measurement_t * tmp2)
```

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

Parameters

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.

Here is the call graph for this function:

6.3.5.7.3.6 __tm_calibration_object_init()

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

Time measurement initialization.

Note

Internal use only.

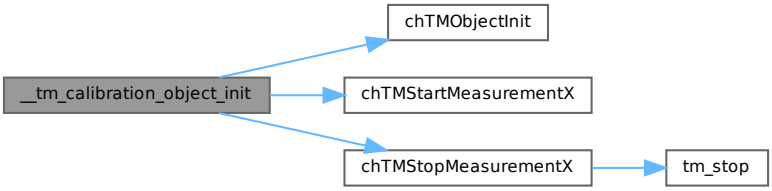
Parameters

out	*tcp*	pointer to the tm_calibration_t structure

Function Class:

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

Here is the call graph for this function:



6.3.6 Synchronization

6.3.6.1 Detailed Description

Synchronization services.

Topics

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

6.3.6.2 Counting Semaphores

6.3.6.2.1 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`.

Data Structures

- struct ch_semaphore

  Semaphore structure.

Macros

- #define sem_insert(qp, tp)
- #define __SEMAPHORE_DATA(name, n)

  Data part of a static semaphore initializer.
- #define SEMAPHORE_DECL(name, n)

  Static semaphore initializer.

Typedefs

- typedef struct ch_semaphore semaphore_t

  Semaphore structure.

Functions

- void chSemObjectInit (semaphore_t *sp, cnt_t n)

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

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

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

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

  Performs a wait operation on a semaphore.

- msg_t chSemWaitTimeout (semaphore_t *sp, sysinterval_t timeout)
  Performs a wait operation on a semaphore with timeout specification.
- msg_t chSemWaitTimeoutS (semaphore_t *sp, sysinterval_t timeout)
  Performs a wait operation on a semaphore with timeout specification.
- void chSemSignal (semaphore_t *sp)
  Performs a signal operation on a semaphore.
- void chSemSignalI (semaphore_t *sp)
  Performs a signal operation on a semaphore.
- void chSemAddCounterI (semaphore_t *sp, cnt_t n)
  Adds the specified value to the semaphore counter.
- msg_t chSemSignalWait (semaphore_t *sps, semaphore_t *spw)
  Performs atomic signal and wait operations on two semaphores.
- static void chSemReset (semaphore_t *sp, cnt_t n)
  Performs a reset operation on the semaphore.
- static void chSemResetI (semaphore_t *sp, cnt_t n)
  Performs a reset operation on the semaphore.
- static void chSemFastWaitI (semaphore_t *sp)
  Decreases the semaphore counter.
- static void chSemFastSignalI (semaphore_t *sp)
  Increases the semaphore counter.
- static cnt_t chSemGetCounterI (const semaphore_t *sp)
  Returns the semaphore counter current value.

6.3.6.2.2 Macro Definition Documentation

6.3.6.2.2.1 sem_insert

```
#define sem_insert(
                qp,
                tp)
```

Value:

```
ch_sch_prio_insert(qp, &tp->hdr.queue)
```

6.3.6.2.2.2 __SEMAPHORE_DATA

```
#define __SEMAPHORE_DATA(
                name,
                n)
```

Value:

```
{__CH_QUEUE_DATA(name.queue), n}
```

Data part of a static semaphore initializer.

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

Parameters

in	*name*	the name of the semaphore variable
in	*n*	the counter initial value, this value must be non-negative

6.3.6.2.2.3 SEMAPHORE_DECL

```
#define SEMAPHORE_DECL(
                name,
                n)
```

Value:

```
semaphore_t name = __SEMAPHORE_DATA(name, n)
```

Static semaphore initializer.

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

Parameters

in	*name*	the name of the semaphore variable
in	*n*	the counter initial value, this value must be non-negative

6.3.6.2.3 Typedef Documentation

6.3.6.2.3.1 semaphore_t

```
typedef struct ch_semaphore semaphore_t
```

Semaphore structure.

6.3.6.2.4 Function Documentation

6.3.6.2.4.1 chSemObjectInit()

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

Initializes a semaphore with the specified counter value.

Parameters

out	*sp*	pointer to a `semaphore_t` structure
in	*n*	initial value of the semaphore counter. Must be non-negative.

Function Class:

Object or module nitializer function.

Here is the call graph for this function:

6.3.6.2.4.2 chSemResetWithMessage()

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

Performs a reset operation on the semaphore.

Postcondition

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

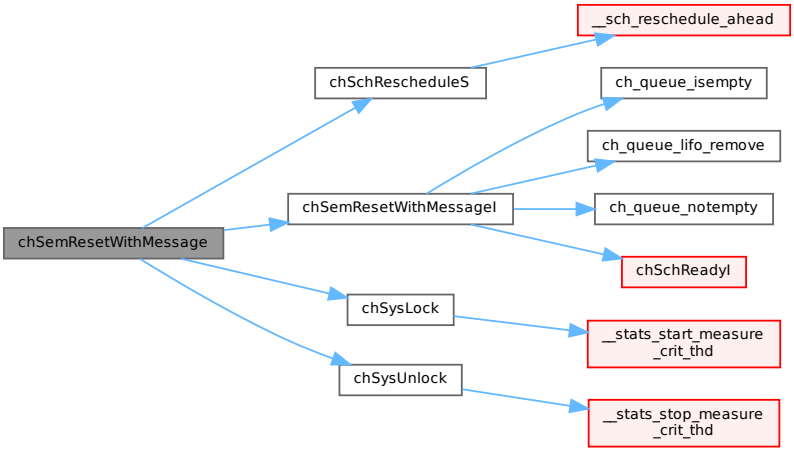
Parameters

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

Function Class:

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

Here is the call graph for this function:



6.3.6.2.4.3 chSemResetWithMessageI()

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

Performs a reset operation on the semaphore.

Postcondition

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

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

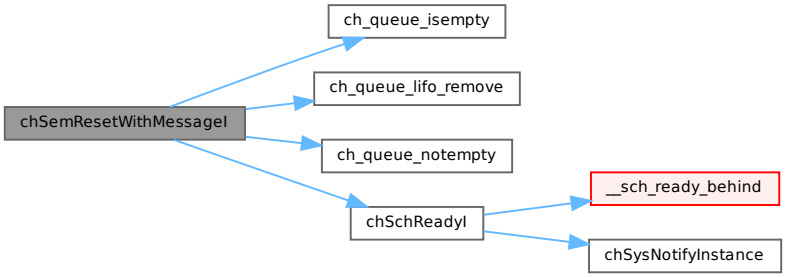
Parameters

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

Function Class:

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

Here is the call graph for this function:



6.3.6.2.4.4 chSemWait()

```
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.

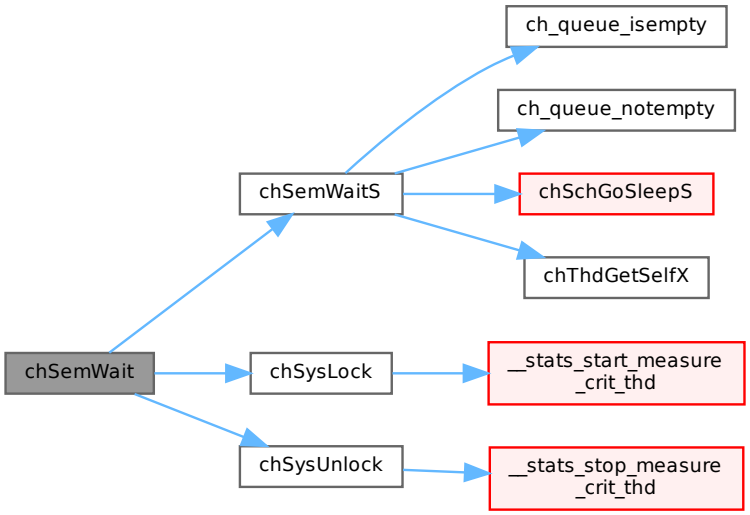
Return values

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.

Here is the call graph for this function:



6.3.6.2.4.5 chSemWaitS()

```
msg_t chSemWaitS (
              semaphore_t * sp)
```

Performs a wait operation on a semaphore.

Parameters

`in`	*sp*	pointer to a `semaphore_t` structure

Returns

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

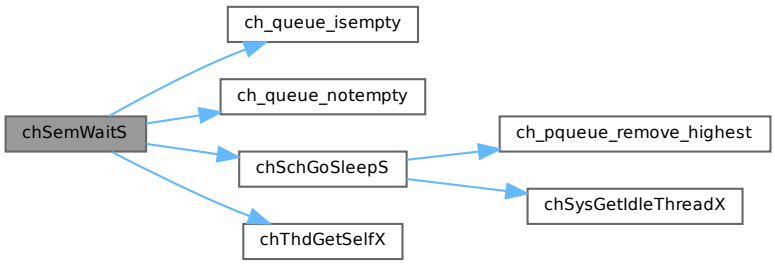
Return values

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:

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

Here is the call graph for this function:



6.3.6.2.4.6 chSemWaitTimeout()

```
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.

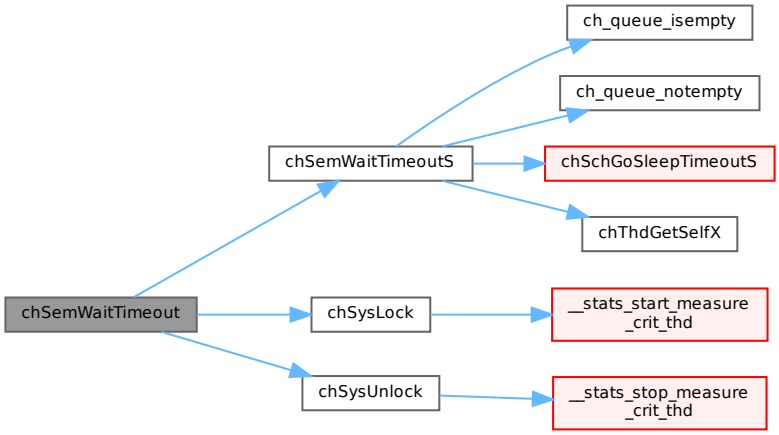
Return values

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.

Function Class:

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

Here is the call graph for this function:



6.3.6.2.4.7 chSemWaitTimeoutS()

```
msg_t chSemWaitTimeoutS (
            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.

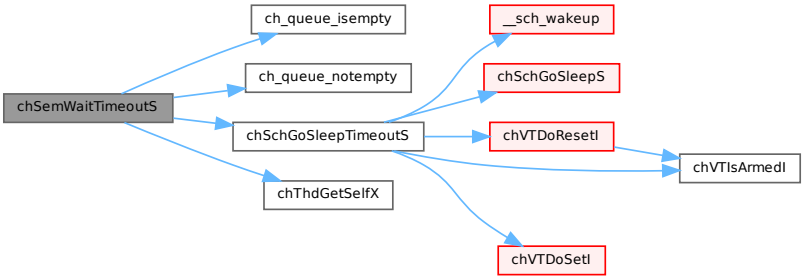
Return values

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.

Function Class:

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

Here is the call graph for this function:



6.3.6.2.4.8 chSemSignal()

```
void chSemSignal (
              semaphore_t * sp)
```

Performs a signal operation on a semaphore.

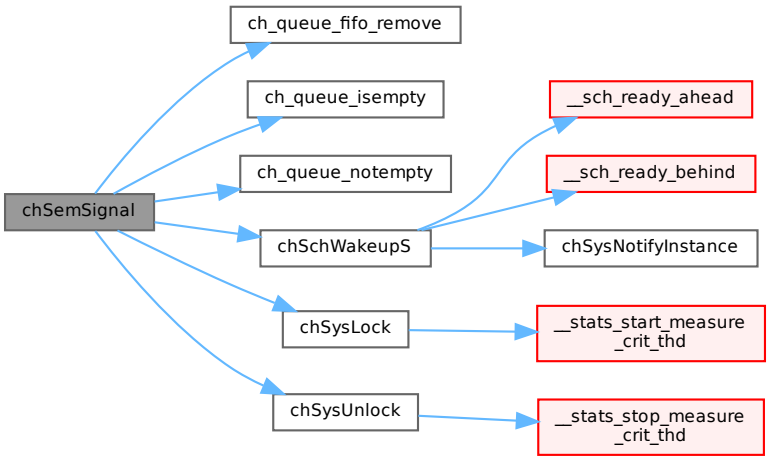
Parameters

in	*sp*	pointer to a semaphore_t structure

Function Class:

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

Here is the call graph for this function:



6.3.6.2.4.9 chSemSignalI()

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

Performs a signal operation on a semaphore.

Postcondition

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

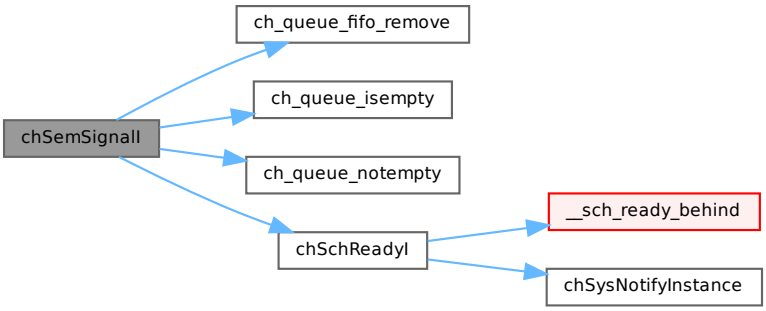
Parameters

in	*sp*	pointer to a semaphore_t structure

Function Class:

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

Here is the call graph for this function:



6.3.6.2.4.10 chSemAddCounterI()

```
void chSemAddCounterI (
            semaphore_t * sp,
            cnt_t n)
```

Adds the specified value to the semaphore counter.

Postcondition

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

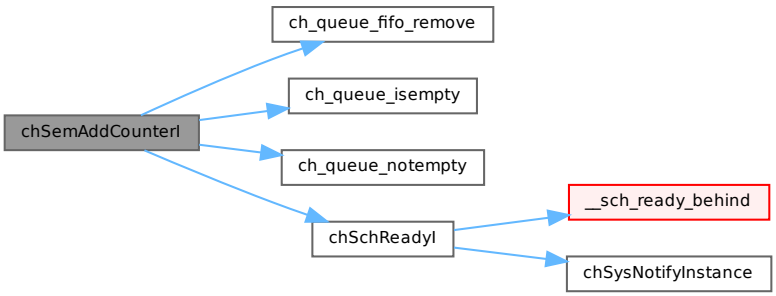
Parameters

in	*sp*	pointer to a semaphore_t structure
in	*n*	value to be added to the semaphore counter. The value must be positive.

Function Class:

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

Here is the call graph for this function:



6.3.6.2.4.11 chSemSignalWait()

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

Performs atomic signal and wait operations on two semaphores.

Parameters

in	*sps*	pointer to a semaphore_t structure to be signaled
in	*spw*	pointer to a semaphore_t structure to wait on

Returns

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

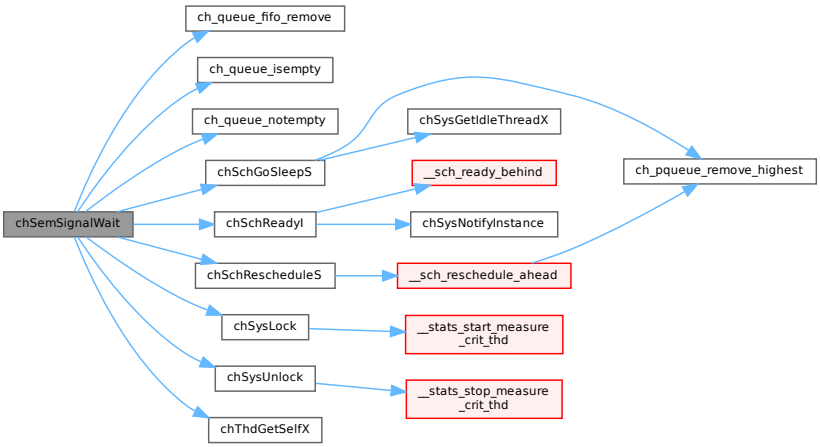
Return values

MSG_OK	if the thread has not stopped on the semaphore or the semaphore has been signaled.
MSG_RESET	if the semaphore has been reset using chSemReset().

Function Class:

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

Here is the call graph for this function:



6.3.6.2.4.12 chSemReset()

```
void chSemReset (
            semaphore_t * sp,
            cnt_t n)  [inline], [static]
```

Performs a reset operation on the semaphore.

Postcondition

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

Note

This function implicitly sends `MSG_RESET` as message.

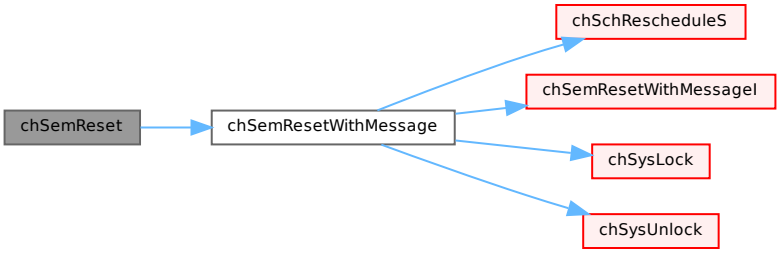
Parameters

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

Function Class:

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

Here is the call graph for this function:



6.3.6.2.4.13 chSemResetI()

```
void chSemResetI (
              semaphore_t * sp,
              cnt_t n)  [inline], [static]
```

Performs a reset operation on the semaphore.

Postcondition

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

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

Note

This function implicitly sends `MSG_RESET` as message.

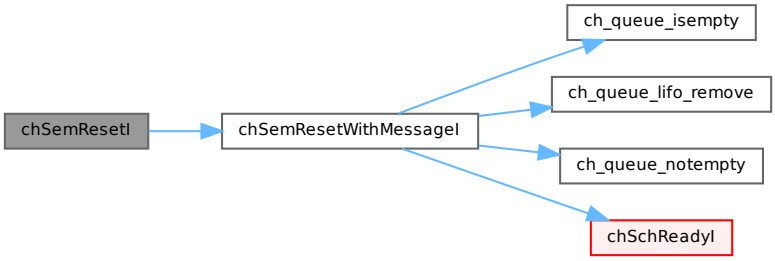
Parameters

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

Function Class:

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

Here is the call graph for this function:



6.3.6.2.4.14 chSemFastWaitI()

```
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.

6.3.6.2.4.15 chSemFastSignalI()

```
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.

6.3.6.2.4.16 chSemGetCounterI()

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

Returns the semaphore counter current value.

Parameters

in	*sp*	pointer to a semaphore_t structure

Returns

The semaphore counter value.

Function Class:

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

6.3.6.3 Mutexes

6.3.6.3.1 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.

Data Structures

- struct ch_mutex

  Mutex structure.

Macros

- #define __MUTEX_DATA(name)

  Data part of a static mutex initializer.
- #define MUTEX_DECL(name)

  Static mutex initializer.

Typedefs

- typedef struct ch_mutex mutex_t

  Type of a mutex structure.

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.
- static bool chMtxQueueNotEmptyS (mutex_t *mp)
  Returns true if the mutex queue contains at least a waiting thread.
- static thread_t * chMtxGetOwnerI (mutex_t *mp)
  Returns the mutex owner thread.
- static mutex_t * chMtxGetNextMutexX (void)
  Returns the next mutex in the mutexes stack of the current thread.

6.3.6.3.2 Macro Definition Documentation

6.3.6.3.2.1 __MUTEX_DATA

```
#define __MUTEX_DATA(
              name)
```

Value:

```
{__CH_QUEUE_DATA(name.queue), NULL, NULL, 0}
```

Data part of a static mutex initializer.

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

Parameters

in	*name*	the name of the mutex variable

6.3.6.3.2.2 MUTEX_DECL

```
#define MUTEX_DECL(
              name)
```

Value:

```
mutex_t name = __MUTEX_DATA(name)
```

Static mutex initializer.

Statically initialized mutexes require no explicit initialization using chMtxInit().

Parameters

in	*name*	the name of the mutex variable

6.3.6.3.3 Typedef Documentation

6.3.6.3.3.1 mutex_t

```
typedef struct ch_mutex mutex_t
```

Type of a mutex structure.

6.3.6.3.4 Function Documentation

6.3.6.3.4.1 chMtxObjectInit()

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

Initializes s `mutex_t` structure.

Parameters

out	*mp*	pointer to a `mutex_t` structure

Function Class:

Object or module nitializer function.

Here is the call graph for this function:

6.3.6.3.4.2 chMtxLock()

```
void chMtxLock (
              mutex_t * mp)
```

Locks the specified mutex.

Postcondition

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

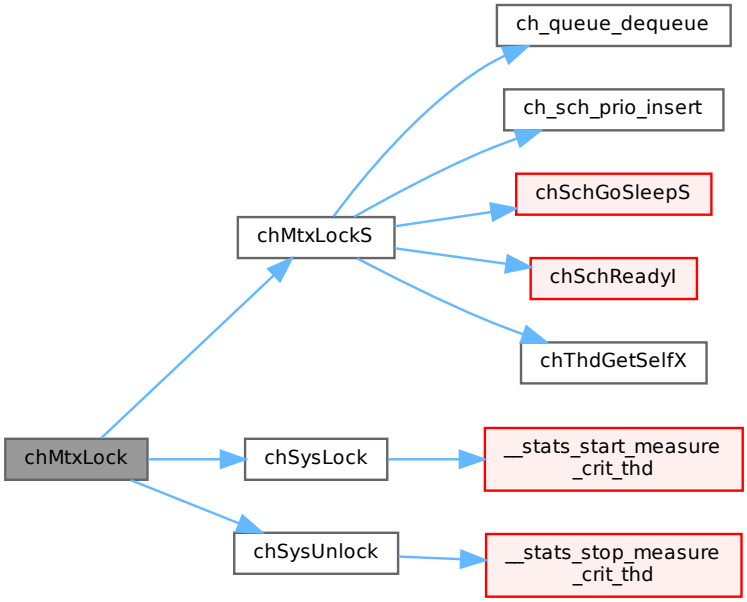
Parameters

in	*mp*	pointer to the mutex_t structure

Function Class:

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

Here is the call graph for this function:



6.3.6.3.4.3 chMtxLockS()

```
void chMtxLockS (
              mutex_t * mp)
```

Locks the specified mutex.

Postcondition

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

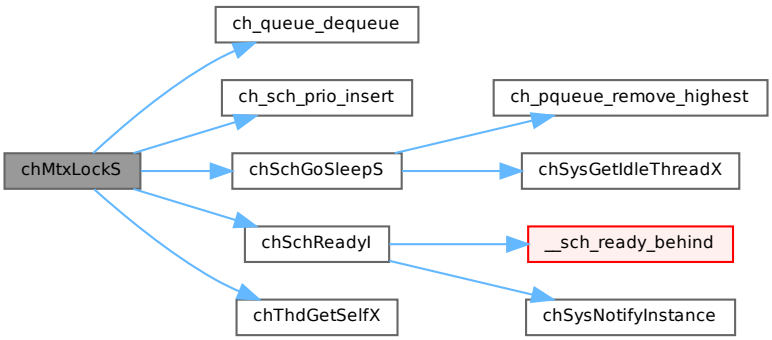
Parameters

in	*mp*	pointer to the mutex_t structure

Function Class:

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

Here is the call graph for this function:



6.3.6.3.4.4 chMtxTryLock()

```
bool chMtxTryLock (
              mutex_t * mp)
```

Tries to lock a mutex.

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

Postcondition

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

Note

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

Parameters

in	*mp*	pointer to the mutex_t structure

Returns

The operation status.

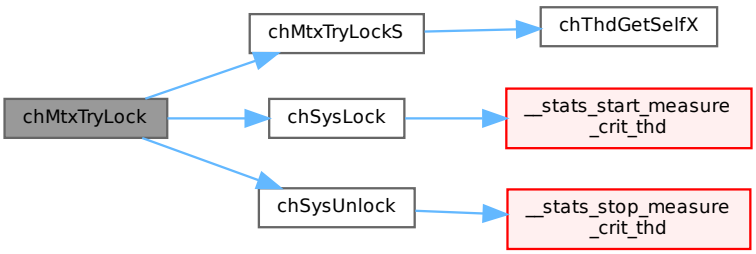
Return values

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

Function Class:

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

Here is the call graph for this function:



6.3.6.3.4.5 chMtxTryLockS()

```
bool chMtxTryLockS (
              mutex_t * mp)
```

Tries to lock a mutex.

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

Postcondition

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

Note

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

Parameters

in	*mp*	pointer to the mutex_t structure

Returns

The operation status.

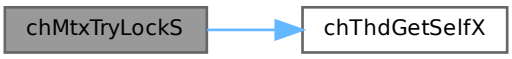
Return values

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

Function Class:

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

Here is the call graph for this function:



6.3.6.3.4.6 chMtxUnlock()

```
void chMtxUnlock (
             mutex_t * mp)
```

Unlocks the specified mutex.

Note

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

Precondition

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

Postcondition

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

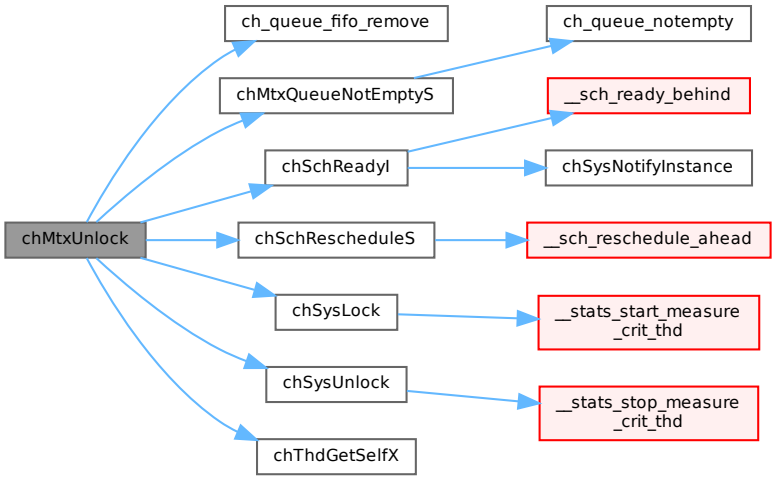
Parameters

in	*mp*	pointer to the mutex_t structure

Function Class:

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

Here is the call graph for this function:



6.3.6.3.4.7 chMtxUnlockS()

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

Unlocks the specified mutex.

Note

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

Precondition

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

Postcondition

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

This function does not reschedule so a call to a rescheduling function must be performed before unlocking the kernel.

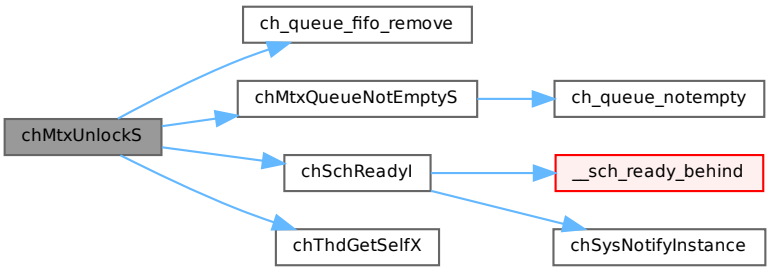
Parameters

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:



6.3.6.3.4.8 chMtxUnlockAllS()

```
void chMtxUnlockAllS (
              void )
```

Unlocks all mutexes owned by the invoking thread.

Postcondition

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

This function does not reschedule so a call to a rescheduling function must be performed before unlocking the kernel.

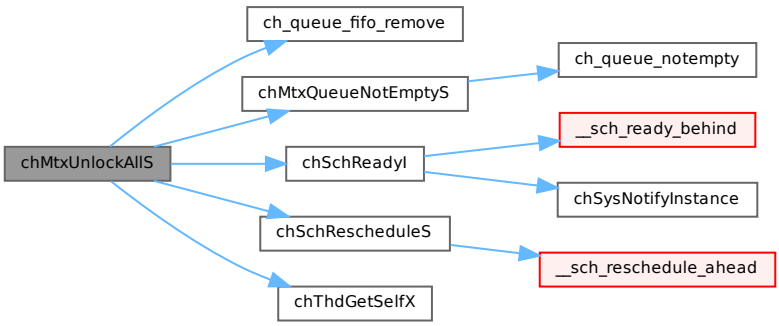
Note

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

Function Class:

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

Here is the call graph for this function:



6.3.6.3.4.9 chMtxUnlockAll()

```
void chMtxUnlockAll (
             void )
```

Unlocks all mutexes owned by the invoking thread.

Postcondition

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

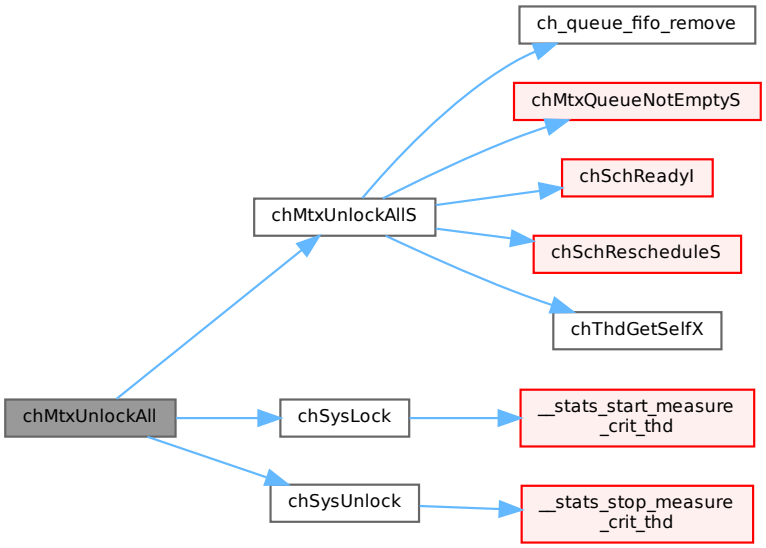
Note

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

Function Class:

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

Here is the call graph for this function:



6.3.6.3.4.10 chMtxQueueNotEmptyS()

```
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:

6.3.6.3.4.11 chMtxGetOwnerI()

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

Returns the mutex owner thread.

Parameters

out	*mp*	pointer to a mutex_t structure

Returns

The owner thread.

Return values

NULL	if the mutex is not owned.

Function Class:

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

6.3.6.3.4.12 chMtxGetNextMutexX()

```
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:

6.3.6.4 Condition Variables

6.3.6.4.1 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`.

Data Structures

- struct condition_variable

  condition_variable_t structure.

Macros

- #define __CONDVAR_DATA(name)

  Data part of a static condition variable initializer.
- #define CONDVAR_DECL(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 s condition_variable_t structure.
- void chCondSignal (condition_variable_t *cp)
  Signals one thread that is waiting on the condition variable.
- void chCondSignalI (condition_variable_t *cp)
  Signals one thread that is waiting on the condition variable.
- void chCondBroadcast (condition_variable_t *cp)
  Signals all threads that are waiting on the condition variable.
- void chCondBroadcastI (condition_variable_t *cp)
  Signals all threads that are waiting on the condition variable.
- msg_t chCondWait (condition_variable_t *cp)
  Waits on the condition variable releasing the mutex lock.
- msg_t chCondWaitS (condition_variable_t *cp)
  Waits on the condition variable releasing the mutex lock.
- msg_t chCondWaitTimeout (condition_variable_t *cp, sysinterval_t timeout)
  Waits on the condition variable releasing the mutex lock.
- msg_t chCondWaitTimeoutS (condition_variable_t *cp, sysinterval_t timeout)
  Waits on the condition variable releasing the mutex lock.

6.3.6.4.2 Macro Definition Documentation

6.3.6.4.2.1 __CONDVAR_DATA

```
#define __CONDVAR_DATA(
              name)
```

Value:

```
{__CH_QUEUE_DATA(name.queue)}
```

Data part of a static condition variable initializer.

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

Parameters

in	*name*	the name of the condition variable

6.3.6.4.2.2 CONDVAR_DECL

```
#define CONDVAR_DECL(
              name)
```

Value:

```
condition_variable_t name = __CONDVAR_DATA(name)
```

Static condition variable initializer.

Statically initialized condition variables require no explicit initialization using `chCondInit()`.

Parameters

in	*name*	the name of the condition variable

6.3.6.4.3 Typedef Documentation

6.3.6.4.3.1 condition_variable_t

```
typedef struct condition_variable condition_variable_t
```

condition_variable_t structure.

6.3.6.4.4 Function Documentation

6.3.6.4.4.1 chCondObjectInit()

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

Initializes s condition_variable_t structure.

Parameters

out	*cp*	pointer to a condition_variable_t structure

Function Class:

Object or module nitializer function.

Here is the call graph for this function:

6.3.6.4.4.2 chCondSignal()

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

Signals one thread that is waiting on the condition variable.

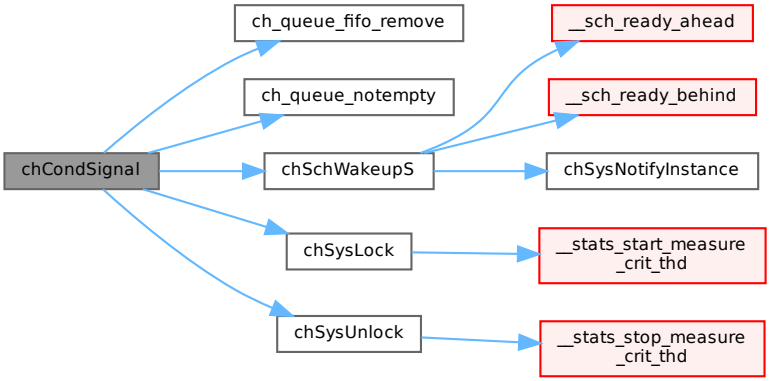
Parameters

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:



6.3.6.4.4.3 chCondSignalI()

```
void chCondSignalI (
            condition_variable_t * cp)
```

Signals one thread that is waiting on the condition variable.

Postcondition

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

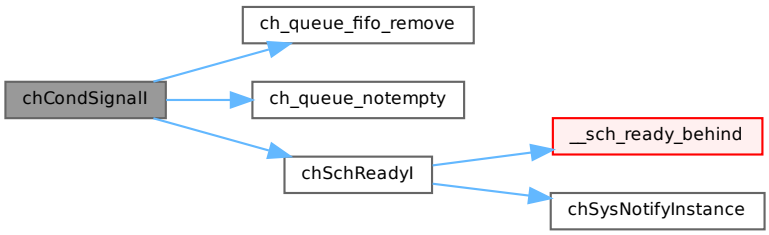
Parameters

in	*cp*	pointer to the condition_variable_t structure

Function Class:

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

Here is the call graph for this function:



6.3.6.4.4.4 chCondBroadcast()

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

Signals all threads that are waiting on the condition variable.

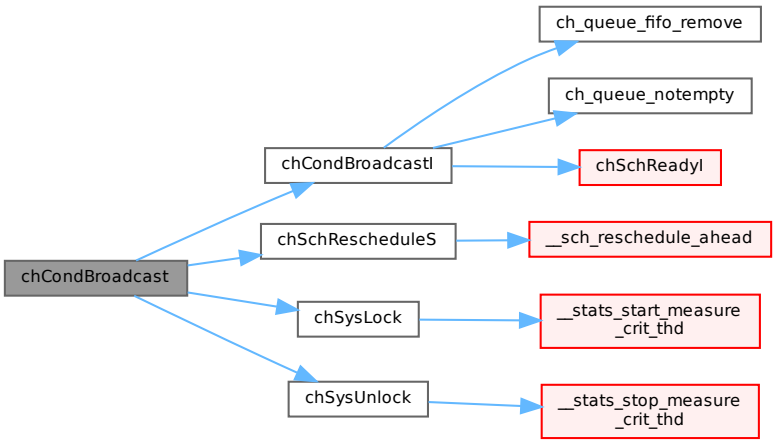
Parameters

in	*cp*	pointer to the condition_variable_t structure

Function Class:

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

Here is the call graph for this function:



6.3.6.4.4.5 chCondBroadcastI()

```
void chCondBroadcastI (
            condition_variable_t * cp)
```

Signals all threads that are waiting on the condition variable.

Postcondition

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

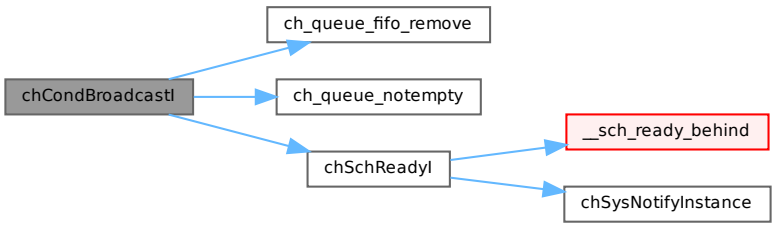
Parameters

in	*cp*	pointer to the condition_variable_t structure

Function Class:

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

Here is the call graph for this function:



6.3.6.4.4.6 chCondWait()

```
msg_t chCondWait (
            condition_variable_t * cp)
```

Waits on the condition variable releasing the mutex lock.

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

Precondition

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

Parameters

in	*cp*	pointer to the condition_variable_t structure

Returns

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

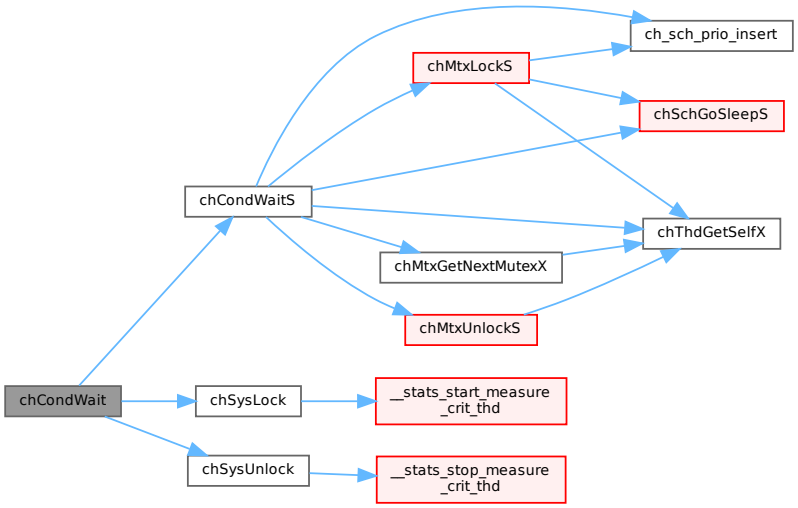
Return values

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

Function Class:

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

Here is the call graph for this function:



6.3.6.4.4.7 chCondWaitS()

```
msg_t chCondWaitS (
            condition_variable_t * cp)
```

Waits on the condition variable releasing the mutex lock.

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

Precondition

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

Parameters

in	*cp*	pointer to the condition_variable_t structure

Returns

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

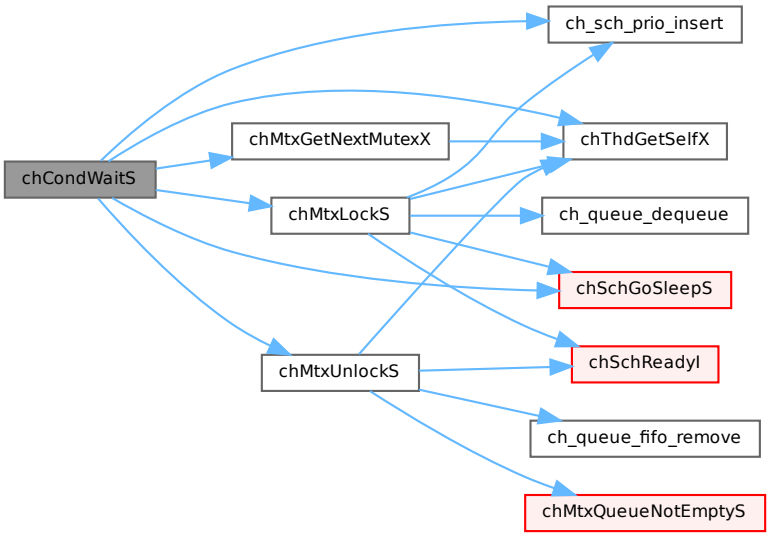
Return values

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

Function Class:

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

Here is the call graph for this function:



6.3.6.4.4.8 chCondWaitTimeout()

```
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

`in`	*cp*	pointer to the `condition_variable_t` structure
`in`	*timeout*	the number of ticks before the operation timeouts, the special values are handled as follow: • *TIME_INFINITE* no timeout. • *TIME_IMMEDIATE* this value is not allowed.

Returns

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

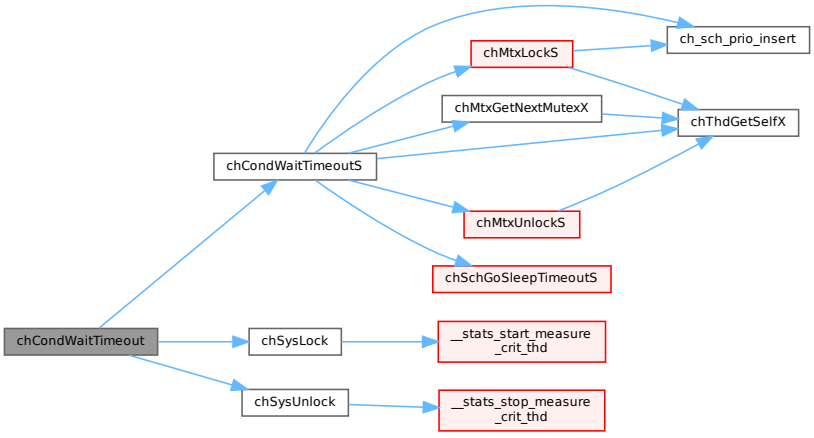
Return values

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

Function Class:

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

Here is the call graph for this function:



6.3.6.4.4.9 chCondWaitTimeoutS()

```
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

in	*cp*	pointer to the `condition_variable_t` structure
in	*timeout*	the number of ticks before the operation timeouts, the special values are handled as follow: • *TIME_INFINITE* no timeout. • *TIME_IMMEDIATE* this value is not allowed.

Returns

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

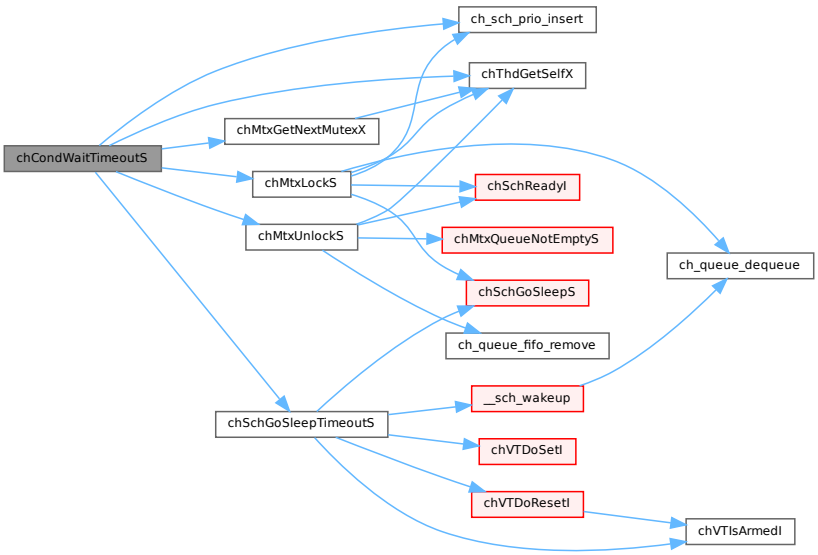
Return values

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

Function Class:

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

Here is the call graph for this function:



6.3.6.5 Event Flags

6.3.6.5.1 Detailed Description

Event Flags, Event Sources and Event Listeners.

Operation mode

Each thread has a mask of pending events inside its `thread_t` structure. Operations defined for events:

- **Wait**, the invoking thread goes to sleep until a certain AND/OR combination of events are signaled.
- **Clear**, a mask of events is cleared from the pending events, the cleared events mask is returned (only the events that were actually pending and then cleared).
- **Signal**, an events mask is directly ORed to the mask of the signaled thread.
- **Broadcast**, each thread registered on an Event Source is signaled with the events specified in its Event Listener.
- **Dispatch**, an events mask is scanned and for each bit set to one an associated handler function is invoked. Bit masks are scanned from bit zero upward.

An Event Source is a special object that can be "broadcasted" by a thread or an interrupt service routine. Broadcasting an Event Source has the effect that all the threads registered on the Event Source will be signaled with an events mask.
An unlimited number of Event Sources can exists in a system and each thread can be listening on an unlimited number of them.

Precondition

In order to use the Events APIs the `CH_CFG_USE_EVENTS` option must be enabled in `chconf.h`.

Postcondition

Enabling events requires 1-4 (depending on the architecture) extra bytes in the `thread_t` structure.

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)

  Returns an event mask from an event identifier.
- #define __EVENTSOURCE_DATA(name)

  Data part of a static event source initializer.
- #define EVENTSOURCE_DECL(name)

  Static event source initializer.
- #define chEvtWaitOne(mask)
- #define chEvtWaitAny(mask)
- #define chEvtWaitAll(mask)

Typedefs

- typedef struct event_listener event_listener_t
- typedef struct event_source event_source_t

  Event Source structure.
- typedef void(∗ evhandler_t) (eventid_t id)

  Event Handler callback function.

Functions

- void chEvtRegisterMaskWithFlagsI (event_source_t ∗esp, event_listener_t ∗elp, eventmask_t events, eventflags_t wflags)

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

  Registers an Event Listener on an Event Source.
- void chEvtUnregister (event_source_t ∗esp, event_listener_t ∗elp)

  Unregisters an Event Listener from its Event Source.
- eventmask_t chEvtGetAndClearEventsI (eventmask_t events)

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

Clears the pending events specified in the events mask.

- eventmask_t chEvtAddEvents (eventmask_t events)

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

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

  Returns the flags associated to an `event_listener_t`.
- void chEvtSignalI (thread_t *tp, eventmask_t events)

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

  Adds a set of event flags directly to the specified `thread_t`.
- void chEvtBroadcastFlagsI (event_source_t *esp, eventflags_t flags)

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

  Signals all the Event Listeners registered on the specified Event Source.
- void chEvtDispatch (const evhandler_t *handlers, eventmask_t events)

  Invokes the event handlers associated to an event flags mask.
- eventmask_t chEvtWaitOne (eventmask_t events)

  Waits for exactly one of the specified events.
- eventmask_t chEvtWaitAny (eventmask_t events)

  Waits for any of the specified events.
- eventmask_t chEvtWaitAll (eventmask_t events)

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

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

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

  Waits for all the specified events.
- static void chEvtObjectInit (event_source_t *esp)

  Initializes an Event Source.
- static void chEvtRegisterMask (event_source_t *esp, event_listener_t *elp, eventmask_t events)

  Registers an Event Listener on an Event Source.
- static void chEvtRegister (event_source_t *esp, event_listener_t *elp, eventid_t event)

  Registers an Event Listener on an Event Source.
- static bool chEvtIsListeningI (event_source_t *esp)

  Verifies if there is at least one `event_listener_t` registered.
- static void chEvtBroadcast (event_source_t *esp)

  Signals all the Event Listeners registered on the specified Event Source.
- static void chEvtBroadcastI (event_source_t *esp)

  Signals all the Event Listeners registered on the specified Event Source.
- static eventmask_t chEvtAddEventsI (eventmask_t events)

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

  Returns the events mask.

6.3.6.5.2 Macro Definition Documentation

6.3.6.5.2.1 ALL_EVENTS

```
#define ALL_EVENTS ((eventmask_t)-1)
```

All events allowed mask.

6.3.6.5.2.2 EVENT_MASK

```
#define EVENT_MASK(
                eid)
```

Value:

```
((eventmask_t)1 << (eventmask_t)(eid))
```

Returns an event mask from an event identifier.

6.3.6.5.2.3 __EVENTSOURCE_DATA

```
#define __EVENTSOURCE_DATA(
                name)
```

Value:

```
{(event_listener_t *)(&name)}
```

Data part of a static event source initializer.

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

Parameters

name	the name of the event source variable

6.3.6.5.2.4 EVENTSOURCE_DECL

```
#define EVENTSOURCE_DECL(
                name)
```

Value:

```
event_source_t name = __EVENTSOURCE_DATA(name)
```

Static event source initializer.

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

Parameters

name	the name of the event source variable

6.3.6.5.2.5 chEvtWaitOne

```
#define chEvtWaitOne(
              mask)
```

Value:

```
chEvtWaitOneTimeout(mask, TIME_INFINITE)
```

6.3.6.5.2.6 chEvtWaitAny

```
#define chEvtWaitAny(
              mask)
```

Value:

```
chEvtWaitAnyTimeout(mask, TIME_INFINITE)
```

6.3.6.5.2.7 chEvtWaitAll

```
#define chEvtWaitAll(
              mask)
```

Value:

```
chEvtWaitAllTimeout(mask, TIME_INFINITE)
```

6.3.6.5.3 Typedef Documentation

6.3.6.5.3.1 event_listener_t

```
typedef struct event_listener event_listener_t
```

6.3.6.5.3.2 event_source_t

```
typedef struct event_source event_source_t
```

Event Source structure.

6.3.6.5.3.3 evhandler_t

```
typedef void(* evhandler_t) (eventid_t id)
```

Event Handler callback function.

6.3.6.5.4 Function Documentation

6.3.6.5.4.1 chEvtRegisterMaskWithFlagsI()

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

Registers an Event Listener on an Event Source.

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

Note

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

Parameters

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

Function Class:

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

Here is the call graph for this function:

6.3.6.5.4.2 chEvtRegisterMaskWithFlags()

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

Registers an Event Listener on an Event Source.

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

Note

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

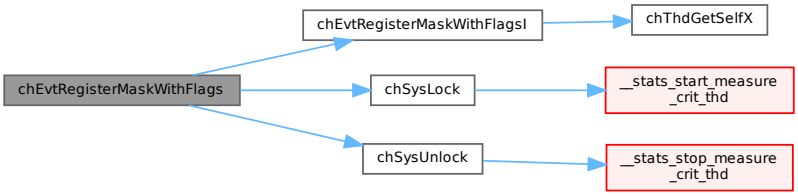
Parameters

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:



6.3.6.5.4.3 chEvtUnregister()

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

Unregisters an Event Listener from its Event Source.

Note

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

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

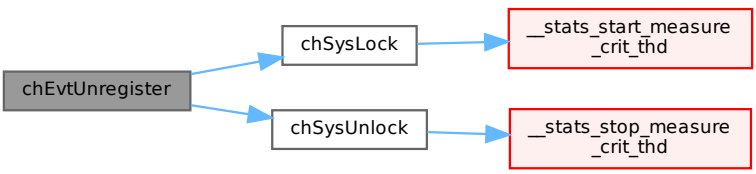
Parameters

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:



6.3.6.5.4.4 chEvtGetAndClearEventsI()

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

Clears the pending events specified in the events mask.

Parameters

in	*events*	the events to be cleared

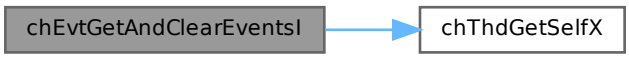
Returns

The mask of pending events that were cleared.

Function Class:

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

Here is the call graph for this function:



6.3.6.5.4.5 chEvtGetAndClearEvents()

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

Clears the pending events specified in the events mask.

Parameters

in	*events*	the events to be cleared

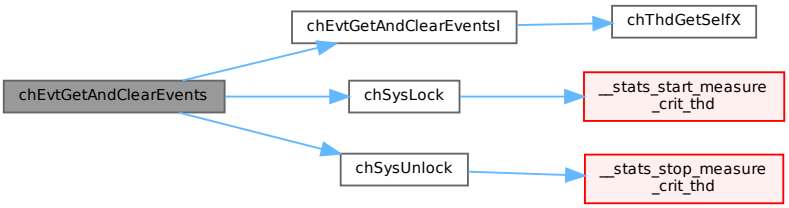
Returns

The mask of pending events that were cleared.

Function Class:

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

Here is the call graph for this function:



6.3.6.5.4.6 chEvtAddEvents()

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

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

Parameters

in	*events*	the events to be added

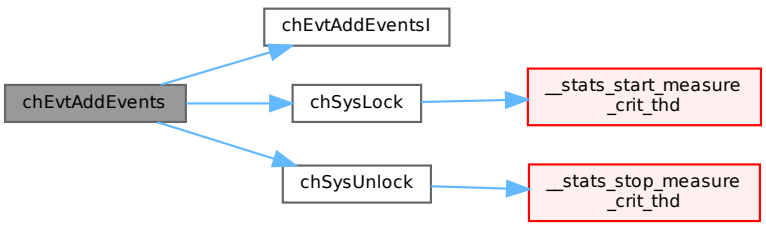
Returns

The mask of currently pending events.

Function Class:

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

Here is the call graph for this function:



6.3.6.5.4.7 chEvtGetAndClearFlagsI()

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

Returns the unmasked flags associated to an event_listener_t.

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

Parameters

in	*elp*	pointer to the event_listener_t structure

Returns

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

Function Class:

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

6.3.6.5.4.8 chEvtGetAndClearFlags()

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

Returns the flags associated to an event_listener_t.

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

Parameters

in	*elp*	pointer to the event_listener_t structure

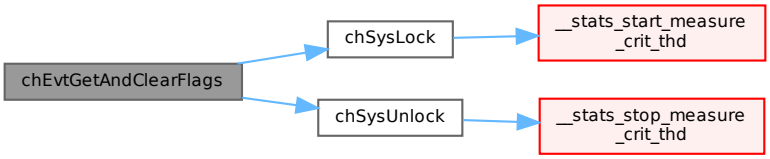
Returns

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

Function Class:

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

Here is the call graph for this function:



6.3.6.5.4.9 chEvtSignalI()

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

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

Postcondition

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

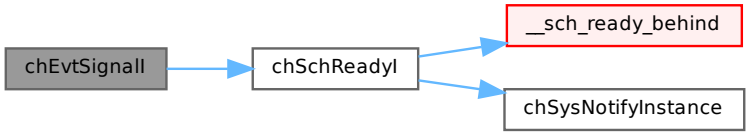
Parameters

in	*tp*	the thread to be signaled
in	*events*	the events set to be ORed

Function Class:

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

Here is the call graph for this function:



6.3.6.5.4.10 chEvtSignal()

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

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

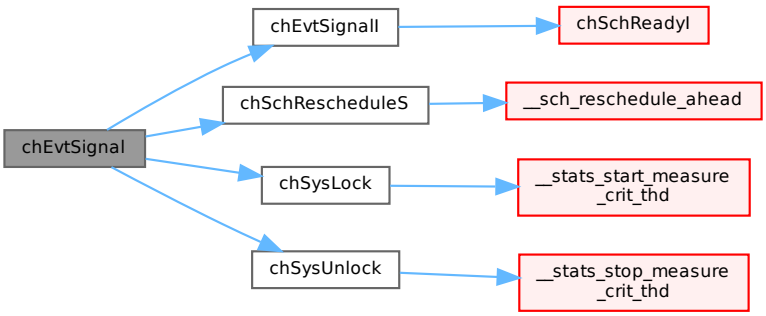
Parameters

in	*tp*	the thread to be signaled
in	*events*	the events set to be ORed

Function Class:

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

Here is the call graph for this function:



6.3.6.5.4.11 chEvtBroadcastFlagsI()

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

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

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

Postcondition

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

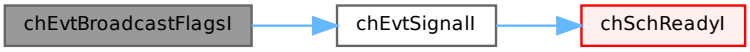
Parameters

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

Function Class:

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

Here is the call graph for this function:



6.3.6.5.4.12 chEvtBroadcastFlags()

```
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.

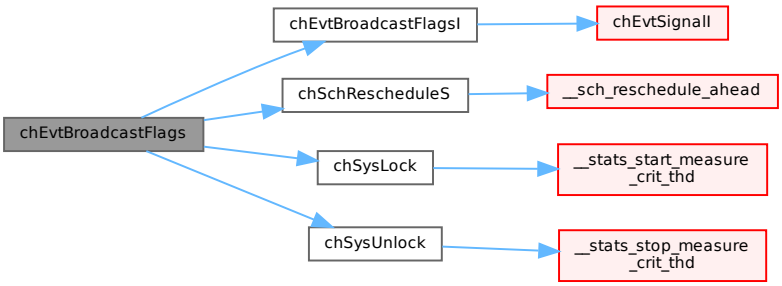
Parameters

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:



6.3.6.5.4.13 chEvtDispatch()

```
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.

6.3.6.5.4.14 chEvtWaitOne()

```
eventmask_t chEvtWaitOne (
             eventmask_t events)
```

Waits for exactly one of the specified events.

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

Note

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

Parameters

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

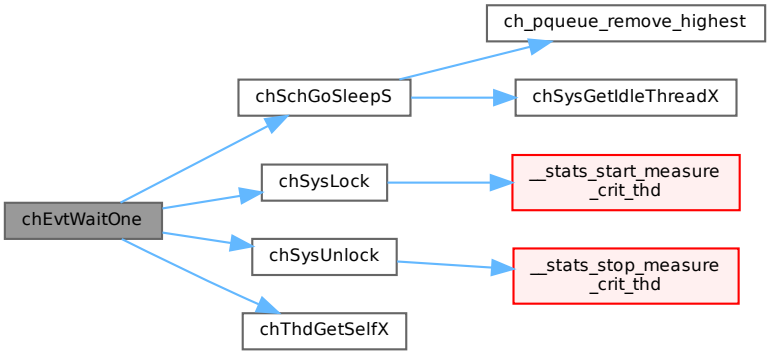
Returns

The mask of the lowest event id served and cleared.

Function Class:

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

Here is the call graph for this function:



6.3.6.5.4.15 chEvtWaitAny()

```
eventmask_t chEvtWaitAny (
            eventmask_t events)
```

Waits for any of the specified events.

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

Parameters

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

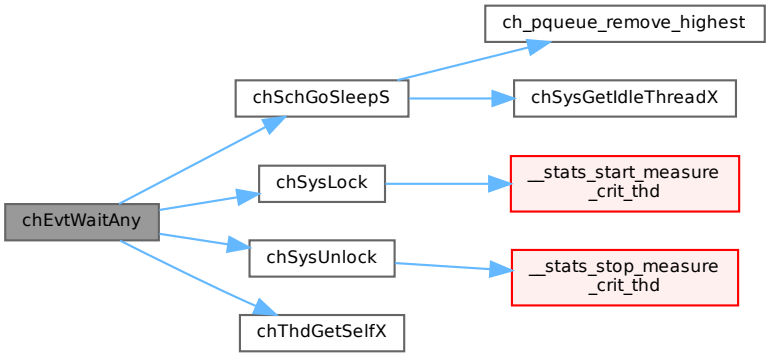
Returns

The mask of the served and cleared events.

Function Class:

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

Here is the call graph for this function:



6.3.6.5.4.16 chEvtWaitAll()

```
eventmask_t chEvtWaitAll (
             eventmask_t events)
```

Waits for all the specified events.

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

Parameters

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

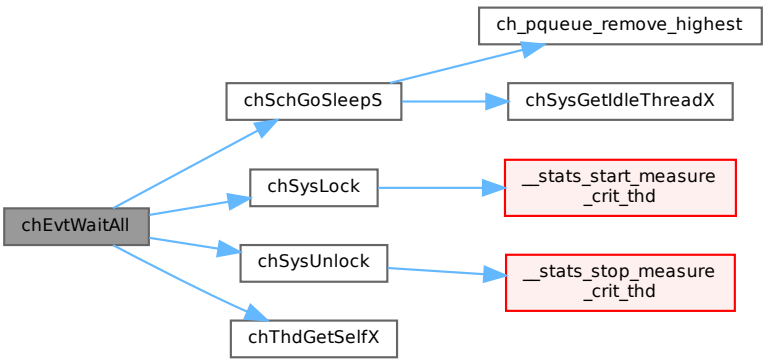
Returns

The mask of the served and cleared events.

Function Class:

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

Here is the call graph for this function:



6.3.6.5.4.17 chEvtWaitOneTimeout()

```
eventmask_t chEvtWaitOneTimeout (
             eventmask_t events,
             sysinterval_t timeout)
```

Waits for exactly one of the specified events.

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

Note

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

Parameters

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

Returns

The mask of the lowest event id served and cleared.

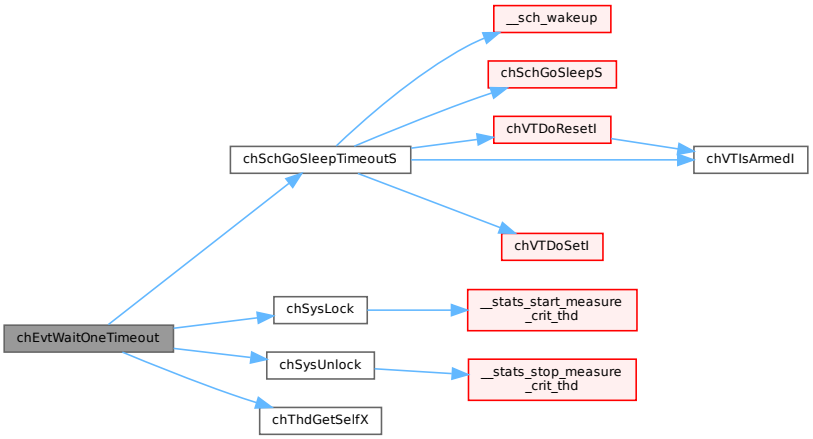
Return values

0	if the operation has timed out.

Function Class:

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

Here is the call graph for this function:



6.3.6.5.4.18 chEvtWaitAnyTimeout()

```
eventmask_t chEvtWaitAnyTimeout (
              eventmask_t events,
              sysinterval_t timeout)
```

Waits for any of the specified events.

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

Parameters

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

Returns

The mask of the served and cleared events.

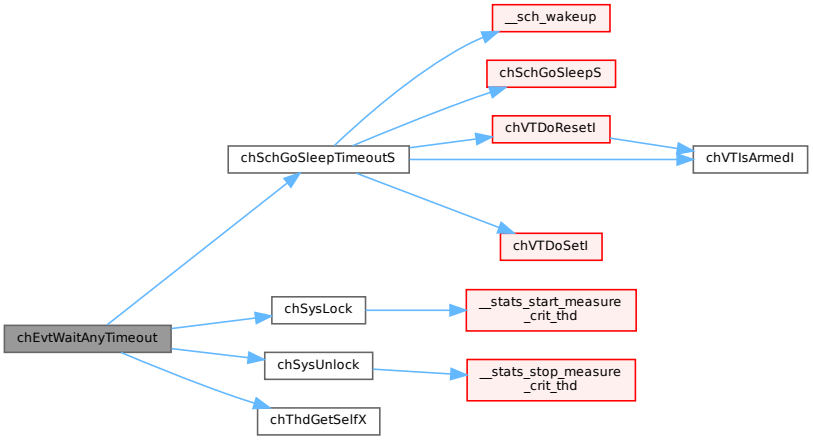
Return values

0	if the operation has timed out.

Function Class:

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

Here is the call graph for this function:



6.3.6.5.4.19 chEvtWaitAllTimeout()

```
eventmask_t chEvtWaitAllTimeout (
              eventmask_t events,
              sysinterval_t timeout)
```

Waits for all the specified events.

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

Parameters

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

Returns

The mask of the served and cleared events.

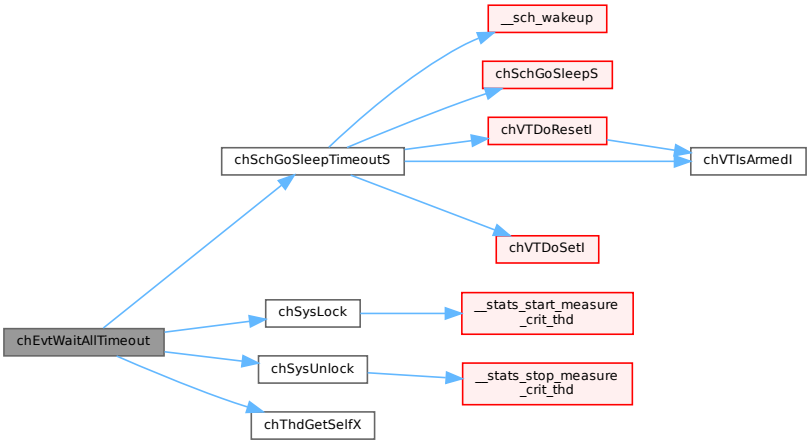
Return values

0	if the operation has timed out.

Function Class:

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

Here is the call graph for this function:



6.3.6.5.4.20 chEvtObjectInit()

```
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:

Object or module nitializer function.

6.3.6.5.4.21 chEvtRegisterMask()

```
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:

6.3.6.5.4.22 chEvtRegister()

```
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.

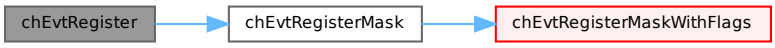
Parameters

in	*esp*	pointer to the event_source_t structure
out	*elp*	pointer to the event_listener_t structure
in	*event*	numeric identifier assigned to the Event Listener. The value must range between zero and the size, in bit, of the eventmask_t type minus one.

Function Class:

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

Here is the call graph for this function:



6.3.6.5.4.23 chEvtIsListeningI()

```
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.

6.3.6.5.4.24 chEvtBroadcast()

```
void chEvtBroadcast (
            event_source_t * esp)  [inline], [static]
```

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

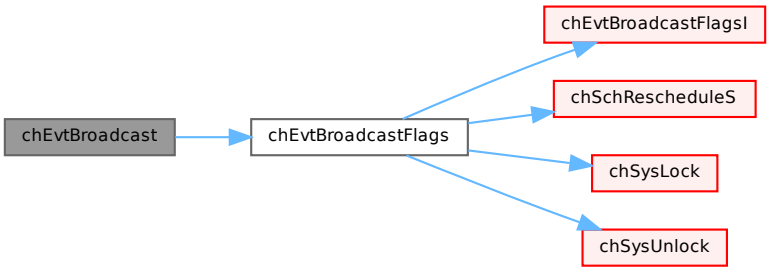
Parameters

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:



6.3.6.5.4.25 chEvtBroadcastI()

```
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.

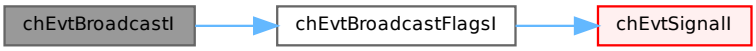
Parameters

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:



6.3.6.5.4.26 chEvtAddEventsI()

```
eventmask_t chEvtAddEventsI (
            eventmask_t events)  [inline], [static]
```

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

Parameters

in	*events*	the events to be added

Returns

The mask of currently pending events.

Function Class:

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

6.3.6.5.4.27 chEvtGetEventsX()

```
eventmask_t chEvtGetEventsX (
            void )  [inline], [static]
```

Returns the events mask.

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

Returns

The pending events mask.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

6.3.6.6 Synchronous Messages

6.3.6.6.1 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.

Macros

- #define __ch_msg_insert(qp, tp)

Functions

- msg_t chMsgSend (thread_t *tp, msg_t msg)

  Sends a message to the specified thread.
- thread_t * chMsgWaitS (void)

  Suspends the thread and waits for an incoming message.
- thread_t * chMsgWaitTimeoutS (sysinterval_t timeout)

  Suspends the thread and waits for an incoming message or a timeout to occur.
- thread_t * chMsgPollS (void)

  Poll to check for an incoming message.
- void chMsgRelease (thread_t *tp, msg_t msg)

  Releases a sender thread specifying a response message.
- static thread_t * chMsgWait (void)

  Suspends the thread and waits for an incoming message.
- static thread_t * chMsgWaitTimeout (sysinterval_t timeout)

  Suspends the thread and waits for an incoming message or a timeout to occur.
- static thread_t * chMsgPoll (void)

  Poll to check for an incoming message.
- static bool chMsgIsPendingI (thread_t *tp)

  Evaluates to `true` if the thread has pending messages.
- static msg_t chMsgGet (thread_t *tp)

  Returns the message carried by the specified thread.
- static void chMsgReleaseS (thread_t *tp, msg_t msg)

  Releases the thread waiting on top of the messages queue.

6.3.6.6.2 Macro Definition Documentation

6.3.6.6.2.1 __ch_msg_insert

```
#define __ch_msg_insert(
              qp,
              tp)
```

Value:

```
ch_sch_prio_insert(qp, &tp->hdr.queue)
```

6.3.6.6.3 Function Documentation

6.3.6.6.3.1 chMsgSend()

```
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

in	*tp*	the pointer to the thread
in	*msg*	the message

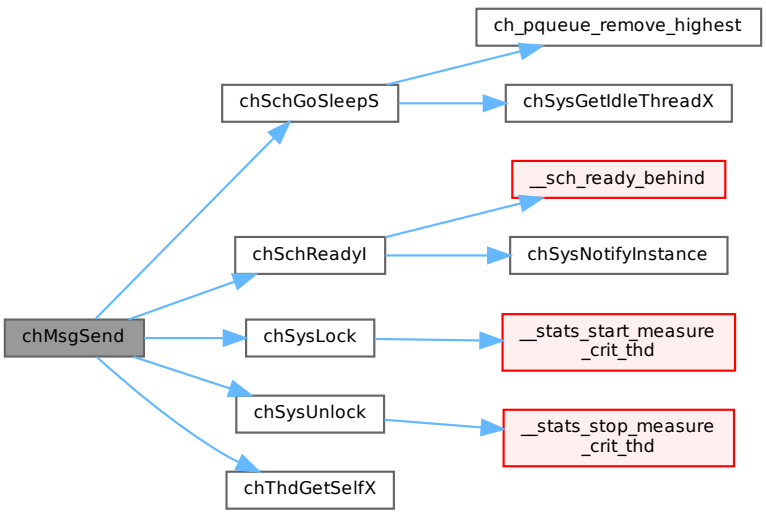
Returns

The answer message from `chMsgRelease()`.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



6.3.6.6.3.2 chMsgWaitS()

```
thread_t * chMsgWaitS (
             void )
```

Suspends the thread and waits for an incoming message.

Postcondition

After receiving a message the function `chMsgGet()` must be called in order to retrieve the message and then `chMsgRelease()` must be invoked in order to acknowledge the reception and send the answer.

Note

If the message is a pointer then you can assume that the data pointed by the message is stable until you invoke `chMsgRelease()` because the sending thread is suspended until then.

The reference counter of the sender thread is not increased, the returned pointer is a temporary reference.

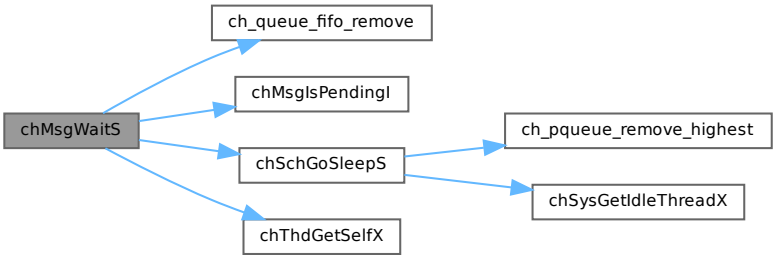
Returns

A pointer to the thread carrying the message.

Function Class:

This is an **S-Class** API, this function can be invoked from within a system lock zone by threads only.

Here is the call graph for this function:



6.3.6.6.3.3 chMsgWaitTimeoutS()

```
thread_t * chMsgWaitTimeoutS (
             sysinterval_t timeout)
```

Suspends the thread and waits for an incoming message or a timeout to occur.

Postcondition

After receiving a message the function `chMsgGet()` must be called in order to retrieve the message and then `chMsgRelease()` must be invoked in order to acknowledge the reception and send the answer.

Note

If the message is a pointer then you can assume that the data pointed by the message is stable until you invoke `chMsgRelease()` because the sending thread is suspended until then.

The reference counter of the sender thread is not increased, the returned pointer is a temporary reference.

Parameters

in	*timeout*	the number of ticks before the operation timeouts, the following special values are allowed: • *TIME_INFINITE* no timeout.

Returns

A pointer to the thread carrying the message.

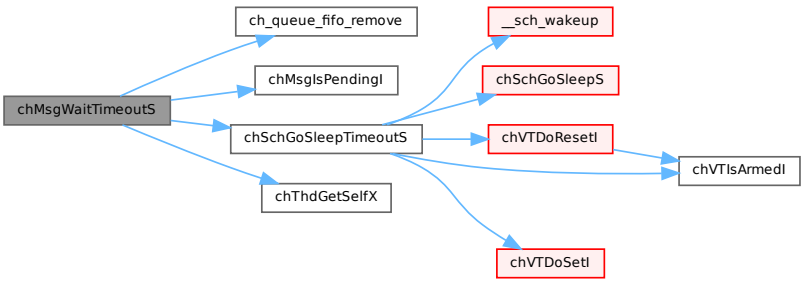
Return values

NULL	if a timeout occurred.

Function Class:

This is an **S-Class** API, this function can be invoked from within a system lock zone by threads only.

Here is the call graph for this function:



6.3.6.6.3.4 chMsgPollS()

```
thread_t * chMsgPollS (
             void )
```

Poll to check for an incoming message.

Postcondition

If a message is available the function `chMsgGet()` must be called in order to retrieve the message and then `chMsgRelease()` must be invoked in order to acknowledge the reception and send the answer.

Note

If the message is a pointer then you can assume that the data pointed by the message is stable until you invoke `chMsgRelease()` because the sending thread is suspended until then.

The reference counter of the sender thread is not increased, the returned pointer is a temporary reference.

Returns

Result of the poll.

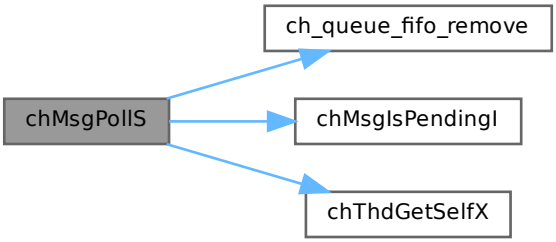
Return values

NULL	if no incoming message waiting.

Function Class:

This is an **S-Class** API, this function can be invoked from within a system lock zone by threads only.

Here is the call graph for this function:



6.3.6.6.3.5 chMsgRelease()

```
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()`.

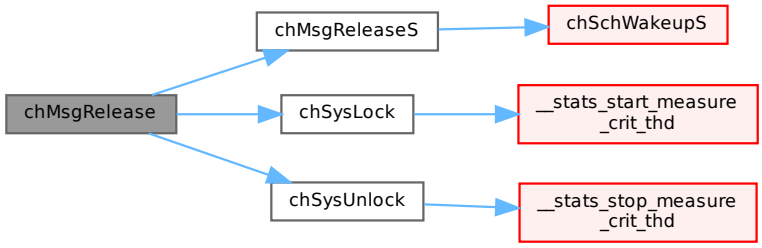
Parameters

in	*tp*	pointer to the thread
in	*msg*	message to be returned to the sender

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



6.3.6.6.3.6 chMsgWait()

```
thread_t * chMsgWait (
             void )  [inline], [static]
```

Suspends the thread and waits for an incoming message.

Postcondition

After receiving a message the function `chMsgGet()` must be called in order to retrieve the message and then `chMsgRelease()` must be invoked in order to acknowledge the reception and send the answer.

Note

If the message is a pointer then you can assume that the data pointed by the message is stable until you invoke `chMsgRelease()` because the sending thread is suspended until then.

The reference counter of the sender thread is not increased, the returned pointer is a temporary reference.

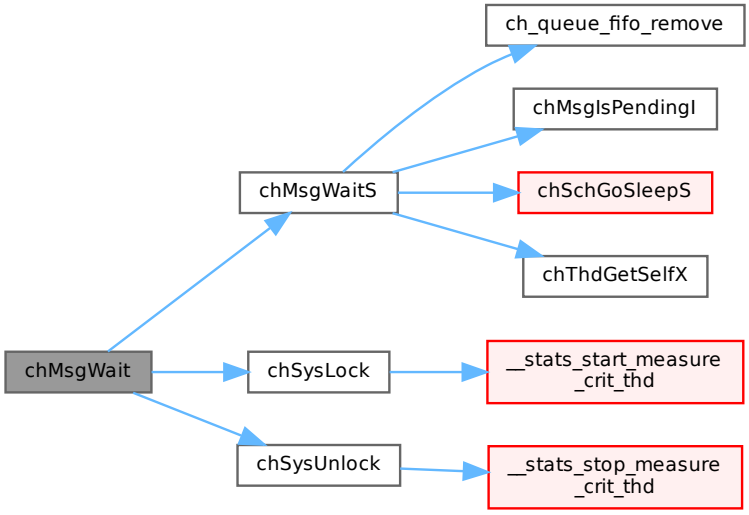
Returns

A pointer to the thread carrying the message.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



6.3.6.6.3.7 chMsgWaitTimeout()

```
thread_t * chMsgWaitTimeout (
             sysinterval_t timeout)  [inline], [static]
```

Suspends the thread and waits for an incoming message or a timeout to occur.

Postcondition

After receiving a message the function `chMsgGet()` must be called in order to retrieve the message and then `chMsgRelease()` must be invoked in order to acknowledge the reception and send the answer.

Note

If the message is a pointer then you can assume that the data pointed by the message is stable until you invoke `chMsgRelease()` because the sending thread is suspended until then.

The reference counter of the sender thread is not increased, the returned pointer is a temporary reference.

Parameters

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 pointer to the thread carrying the message.

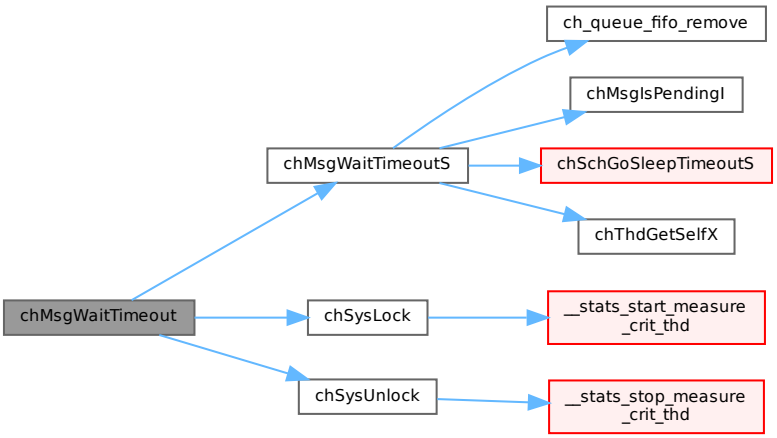
Return values

NULL	if a timeout occurred.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



6.3.6.6.3.8 chMsgPoll()

```
thread_t * chMsgPoll (
             void )  [inline], [static]
```

Poll to check for an incoming message.

Postcondition

If a message is available the function `chMsgGet()` must be called in order to retrieve the message and then `chMsgRelease()` must be invoked in order to acknowledge the reception and send the answer.

Note

If the message is a pointer then you can assume that the data pointed by the message is stable until you invoke `chMsgRelease()` because the sending thread is suspended until then.

The reference counter of the sender thread is not increased, the returned pointer is a temporary reference.

Returns

A pointer to the thread carrying the message.

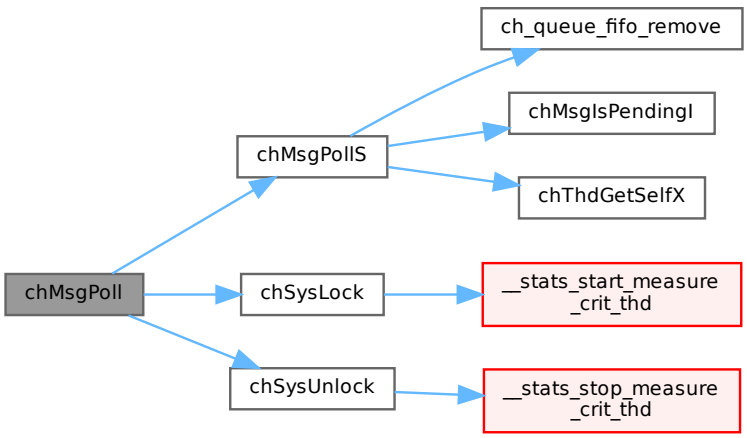
Return values

NULL	if no incoming message waiting.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



6.3.6.6.3.9 chMsgIsPendingI()

```
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.

6.3.6.6.3.10 chMsgGet()

```
msg_t chMsgGet (
              thread_t * tp)  [inline], [static]
```

Returns the message carried by the specified thread.

Precondition

This function must be invoked immediately after exiting a call to chMsgWait().

Parameters

in	*tp*	pointer to the thread

Returns

The message carried by the sender.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

6.3.6.6.3.11 chMsgReleaseS()

```
void chMsgReleaseS (
              thread_t * tp,
              msg_t msg)  [inline], [static]
```

Releases the thread waiting on top of the messages queue.

Precondition

Invoke this function only after a message has been received using chMsgWait().

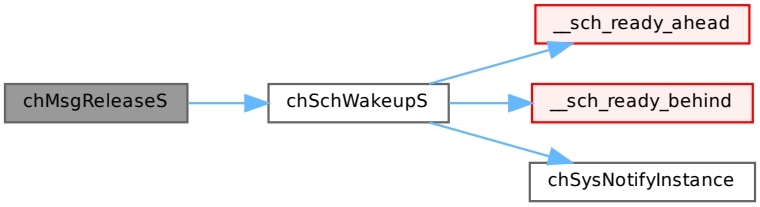
Parameters

in	*tp*	pointer to the thread
in	*msg*	message to be returned to the sender

Function Class:

This is an **S-Class** API, this function can be invoked from within a system lock zone by threads only.

Here is the call graph for this function:



6.3.7 Dynamic Threads

6.3.7.1 Detailed Description

Dynamic threads related APIs and services.

Functions

- thread_t * chThdCreateFromHeap (memory_heap_t *heapp, size_t size, const char *name, tprio_t prio, tfunc_t pf, void *arg)

  Creates a new thread allocating the memory from the heap.
- thread_t * chThdCreateFromMemoryPool (memory_pool_t *mp, const char *name, tprio_t prio, tfunc_t pf, void *arg)

  Creates a new thread allocating the memory from the specified memory pool.

6.3.7.2 Function Documentation

6.3.7.2.1 chThdCreateFromHeap()

```
thread_t * chThdCreateFromHeap (
            memory_heap_t * heapp,
            size_t size,
            const char * name,
            tprio_t prio,
            tfunc_t pf,
            void * arg)
```

Creates a new thread allocating the memory from the heap.

Precondition

The configuration options `CH_CFG_USE_DYNAMIC` and `CH_CFG_USE_HEAP` must be enabled in order to use this function.

Note

A thread can terminate by calling `chThdExit()` or by simply returning from its main function.

The allocated thread memory is not released automatically, it is caller responsibility to call `chThdRelease()` or `chThdWait()` in order to release the allocated memory.

Parameters

`in`	*heapp*	heap from which allocate the memory or `NULL` for the default heap
`in`	*size*	size of the working area to be allocated
`in`	*name*	thread name
`in`	*prio*	the priority level for the new thread
`in`	*pf*	the thread function
`in`	*arg*	an argument passed to the thread function. It can be `NULL`.

Returns

The pointer to the `thread_t` structure allocated for the thread into the working space area.

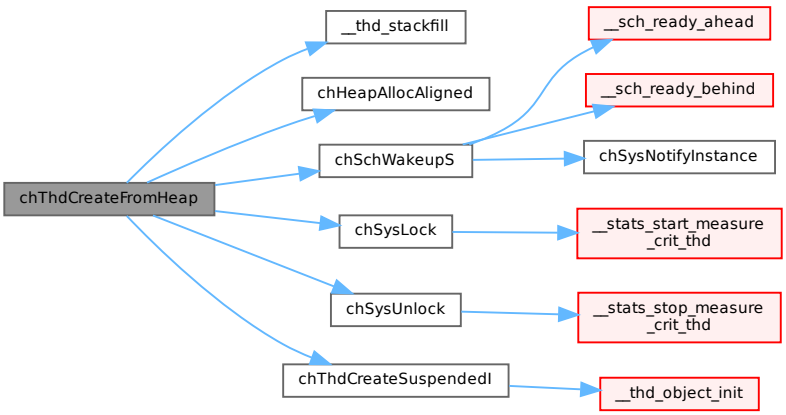
Return values

NULL	if the memory cannot be allocated.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



6.3.7.2.2 chThdCreateFromMemoryPool()

```
thread_t * chThdCreateFromMemoryPool (
             memory_pool_t * mp,
             const char * name,
             tprio_t prio,
             tfunc_t pf,
             void * arg)
```

Creates a new thread allocating the memory from the specified memory pool.

Precondition

The configuration options CH_CFG_USE_DYNAMIC and CH_CFG_USE_MEMPOOLS must be enabled in order to use this function.

The pool must be initialized to contain only objects with alignment PORT_WORKING_AREA_ALIGN.

Note

A thread can terminate by calling chThdExit() or by simply returning from its main function.

The allocated thread memory is not released automatically, it is caller responsibility to call chThdRelease() or chThdWait() in order to release the allocated memory.

Parameters

in	*mp*	pointer to the memory pool object
in	*name*	thread name
in	*prio*	the priority level for the new thread
in	*pf*	the thread function
in	*arg*	an argument passed to the thread function. It can be NULL.

Returns

The pointer to the thread_t structure allocated for the thread into the working space area.

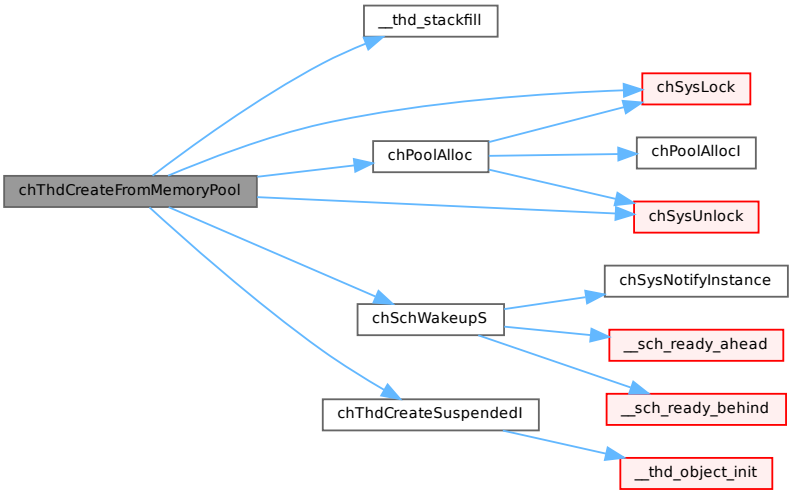
Return values

NULL	if the memory pool is empty.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



6.3.8 Registry

6.3.8.1 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`.

Data Structures

- struct chdebug_t

  ChibiOS/RT memory signature record.

Macros

- #define REG_HEADER(oip)

  Access to the registry list header.
- #define REG_REMOVE(tp)

  Removes a thread from the registry list.
- #define REG_INSERT(oip, tp)

  Adds a thread to the registry list.

Functions

- thread_t * chRegFirstThread (void)

  Returns the first thread in the system.
- thread_t * chRegNextThread (thread_t *tp)

  Returns the thread next to the specified one.
- thread_t * chRegFindThreadByName (const char *name)

  Retrieves a thread pointer by name.
- thread_t * chRegFindThreadByPointer (thread_t *tp)

  Confirms that a pointer is a valid thread pointer.
- thread_t * chRegFindThreadByWorkingArea (stkalign_t *wa)

  Confirms that a working area is being used by some active thread.
- static void __reg_object_init (registry_t *rp)

  Initializes a registry.
- static void chRegSetThreadName (const char *name)

  Sets the current thread name.
- static const char * chRegGetThreadNameX (thread_t *tp)

  Returns the name of the specified thread.
- static void chRegSetThreadNameX (thread_t *tp, const char *name)

  Changes the name of the specified thread.

Variables

- ROMCONST chdebug_t ch_debug
- ROMCONST chdebug_t ch_debug

6.3.8.2 Macro Definition Documentation

6.3.8.2.1 REG_HEADER

```
#define REG_HEADER(
                oip)
```

Value:

```
(&ch_system.reglist.queue)
```

Access to the registry list header.

6.3.8.2.2 REG_REMOVE

```
#define REG_REMOVE(
              tp)
```

Value:

```
(void) ch_queue_dequeue(&(tp)->rqueue)
```

Removes a thread from the registry list.

Note

This macro is not meant for use in application code.

Parameters

in	*tp*	thread to remove from the registry

6.3.8.2.3 REG_INSERT

```
#define REG_INSERT(
              oip,
              tp)
```

Value:

```
ch_queue_insert(REG_HEADER(oip), &(tp)->rqueue)
```

Adds a thread to the registry list.

Note

This macro is not meant for use in application code.

Parameters

in	*oip*	pointer to the OS instance
in	*tp*	thread to add to the registry

6.3.8.3 Function Documentation

6.3.8.3.1 chRegFirstThread()

```
thread_t * chRegFirstThread (
             void )
```

Returns the first thread in the system.

Returns the most ancient thread in the system, usually this is the main thread unless it terminated. A reference is added to the returned thread in order to make sure its status is not lost.

Note

This function cannot return `NULL` because there is always at least one thread in the system.

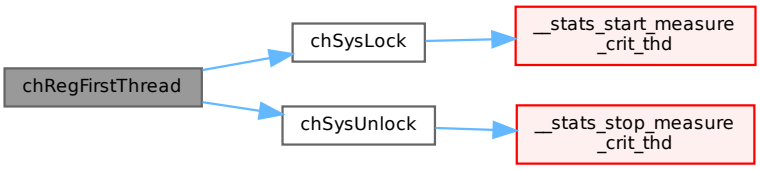
Returns

A reference to the most ancient thread.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



6.3.8.3.2 chRegNextThread()

```
thread_t * chRegNextThread (
             thread_t * tp)
```

Returns the thread next to the specified one.

The reference counter of the specified thread is decremented and the reference counter of the returned thread is incremented.

Parameters

`in`	*tp*	pointer to the thread

Returns

A reference to the next thread.

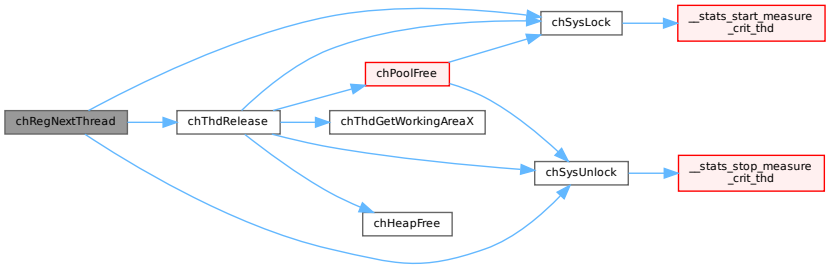
Return values

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:



6.3.8.3.3 chRegFindThreadByName()

```
thread_t * chRegFindThreadByName (
              const char * name)
```

Retrieves a thread pointer by name.

Note

The reference counter of the found thread is increased by one so it cannot be disposed incidentally after the pointer has been returned.

Parameters

in	*name*	the thread name

Returns

A pointer to the found thread.

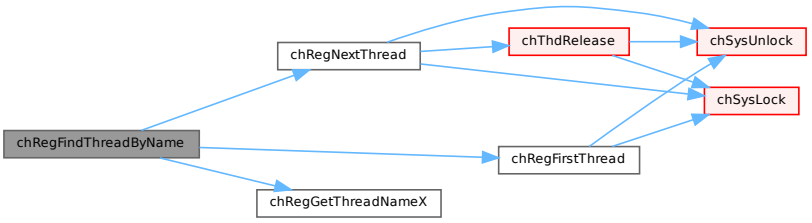
Return values

NULL	if a matching thread has not been found.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



6.3.8.3.4 chRegFindThreadByPointer()

```
thread_t * chRegFindThreadByPointer (
            thread_t * tp)
```

Confirms that a pointer is a valid thread pointer.

Note

The reference counter of the found thread is increased by one so it cannot be disposed incidentally after the pointer has been returned.

Parameters

in	*tp*	pointer to the thread

Returns

A pointer to the found thread.

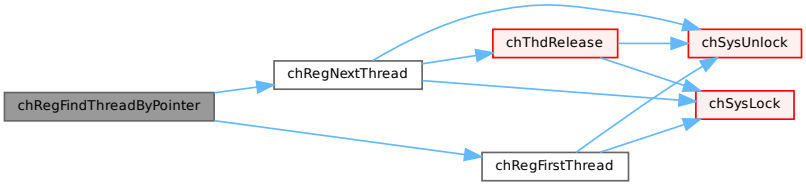
Return values

NULL	if a matching thread has not been found.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



6.3.8.3.5 chRegFindThreadByWorkingArea()

```
thread_t * chRegFindThreadByWorkingArea (
            stkalign_t * wa)
```

Confirms that a working area is being used by some active thread.

Note

The reference counter of the found thread is increased by one so it cannot be disposed incidentally after the pointer has been returned.

Parameters

in	*wa*	pointer to a static working area

Returns

A pointer to the found thread.

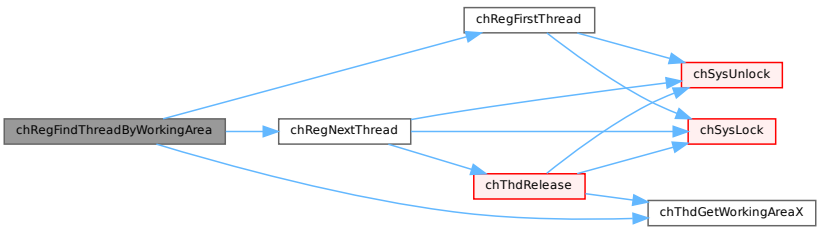
Return values

NULL	if a matching thread has not been found.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



6.3.8.3.6 __reg_object_init()

```
void __reg_object_init (
            registry_t * rp)  [inline], [static]
```

Initializes a registry.

Note

Internal use only.

Parameters

out	*rp*	pointer to a registry_t structure

Function Class:

Object or module nitializer function.

Here is the call graph for this function:

6.3.8.3.7 chRegSetThreadName()

```
void chRegSetThreadName (
             const char * name)  [inline], [static]
```

Sets the current thread name.

Precondition

This function only stores the pointer to the name if the option CH_CFG_USE_REGISTRY is enabled else no action is performed.

Parameters

in	*name*	thread name as a zero terminated string

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

6.3.8.3.8 chRegGetThreadNameX()

```
const char * chRegGetThreadNameX (
             thread_t * tp)  [inline], [static]
```

Returns the name of the specified thread.

Precondition

This function only returns the pointer to the name if the option CH_CFG_USE_REGISTRY is enabled else NULL is returned.

Parameters

in	*tp*	pointer to the thread

Returns

Thread name as a zero terminated string.

Return values

NULL	if the thread name has not been set.

6.3.8.3.9 chRegSetThreadNameX()

```
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

in	*tp*	pointer to the thread
in	*name*	thread name as a zero terminated string

Function Class:

This is an **X-Class** API, this function can be invoked from any context.

6.3.8.4 Variable Documentation

6.3.8.4.1 ch_debug [1/2]

```
ROMCONST chdebug_t ch_debug
```

6.3.8.4.2 ch_debug [2/2]

```
ROMCONST chdebug_t ch_debug  [extern]
```

6.3.9 Debug

6.3.9.1 Detailed Description

Topics

- Checks and Assertions
- Tracing
- Statistics

6.3.9.2 Checks and Assertions

6.3.9.2.1 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.

Data Structures

- struct ch_system_debug

  System debug data structure.

Macros

- #define __dbg_check_disable()
- #define __dbg_check_suspend()
- #define __dbg_check_enable()
- #define __dbg_check_lock()
- #define __dbg_check_unlock()
- #define __dbg_check_lock_from_isr()
- #define __dbg_check_unlock_from_isr()
- #define __dbg_check_enter_isr()
- #define __dbg_check_leave_isr()
- #define chDbgCheckClassI()
- #define chDbgCheckClassS()

Typedefs

- typedef struct ch_system_debug system_debug_t

  System debug data structure.

Functions

- void __dbg_check_disable (void)
  Guard code for `chSysDisable()`.
- void __dbg_check_suspend (void)
  Guard code for `chSysSuspend()`.
- void __dbg_check_enable (void)
  Guard code for `chSysEnable()`.
- void __dbg_check_lock (void)
  Guard code for `chSysLock()`.
- void __dbg_check_unlock (void)
  Guard code for `chSysUnlock()`.
- void __dbg_check_lock_from_isr (void)
  Guard code for `chSysLockFromIsr()`.
- void __dbg_check_unlock_from_isr (void)
  Guard code for `chSysUnlockFromIsr()`.
- void __dbg_check_enter_isr (void)
  Guard code for `CH_IRQ_PROLOGUE()`.
- void __dbg_check_leave_isr (void)
  Guard code for `CH_IRQ_EPILOGUE()`.
- void chDbgCheckClassI (void)
  I-class functions context check.
- void chDbgCheckClassS (void)
  S-class functions context check.
- static void __dbg_object_init (system_debug_t *sdp)
  Debug support initialization.

6.3.9.2.2 Macro Definition Documentation

6.3.9.2.2.1 __dbg_check_disable

```
void __dbg_check_disable()
```

6.3.9.2.2.2 __dbg_check_suspend

```
void __dbg_check_suspend()
```

6.3.9.2.2.3 __dbg_check_enable

```
void __dbg_check_enable()
```

6.3.9.2.2.4 __dbg_check_lock

```
void __dbg_check_lock()
```

6.3.9.2.2.5 __dbg_check_unlock

```
void __dbg_check_unlock()
```

6.3.9.2.2.6 __dbg_check_lock_from_isr

```
void __dbg_check_lock_from_isr()
```

6.3.9.2.2.7 __dbg_check_unlock_from_isr

```
void __dbg_check_unlock_from_isr()
```

6.3.9.2.2.8 __dbg_check_enter_isr

```
void __dbg_check_enter_isr()
```

6.3.9.2.2.9 __dbg_check_leave_isr

```
void __dbg_check_leave_isr()
```

6.3.9.2.2.10 chDbgCheckClassI

```
void chDbgCheckClassI()
```

6.3.9.2.2.11 chDbgCheckClassS

```
void chDbgCheckClassS()
```

6.3.9.2.2.12 CH_DBG_STACK_FILL_VALUE

```
#define CH_DBG_STACK_FILL_VALUE 0x55
```

Fill value for thread stack area in debug mode.

6.3.9.2.2.13 chDbgCheck

```
#define chDbgCheck(
              c)
```

Value:

```
  do {                                                      \
  /*lint -save -e506 -e774 [2.1, 14.3] Can be a constant by design.*/       \
  if (CH_DBG_ENABLE_CHECKS != FALSE) {                                      \
    if (unlikely(!(c))) {                                                   \
  /*lint -restore*/                                                        \
      chSysHalt(__func__);                                                  \
    }                                                                       \
  }                                                                         \
} while (false)
```

Function parameters check.

If the condition check fails then the kernel panics and halts.

Note

The condition is tested only if the CH_DBG_ENABLE_CHECKS switch is specified in chconf.h else the macro does nothing.

Parameters

in	*c*	the condition to be verified to be true

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

6.3.9.2.2.14 chDbgAssert

```
#define chDbgAssert(
                c,
                r)
```

Value:

```
  do {                                                      \
  /*lint -save -e506 -e774 [2.1, 14.3] Can be a constant by design.*/       \
  if (CH_DBG_ENABLE_ASSERTS != FALSE) {                                     \
    if (unlikely(!(c))) {                                                   \
  /*lint -restore*/                                                        \
      chSysHalt(__func__);                                                  \
    }                                                                       \
  }                                                                         \
} while (false)
```

Condition assertion.

If the condition check fails then the kernel panics with a message and halts.

Note

The condition is tested only if the CH_DBG_ENABLE_ASSERTS switch is specified in chconf.h else the macro does nothing.

The remark string is not currently used except for putting a comment in the code about the assertion.

Parameters

in	*c*	the condition to be verified to be true
in	*r*	a remark string

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

6.3.9.2.3 Typedef Documentation

6.3.9.2.3.1 system_debug_t

```
typedef struct ch_system_debug system_debug_t
```

System debug data structure.

6.3.9.2.4 Function Documentation

6.3.9.2.4.1 __dbg_check_disable()

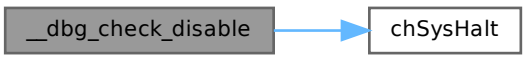
```
void __dbg_check_disable (
              void )
```

Guard code for chSysDisable().

Function Class:

Not an API, this function is for internal use only.

Here is the call graph for this function:



6.3.9.2.4.2 __dbg_check_suspend()

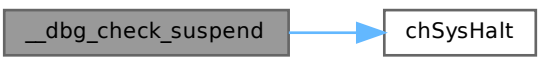
```
void __dbg_check_suspend (
              void )
```

Guard code for chSysSuspend().

Function Class:

Not an API, this function is for internal use only.

Here is the call graph for this function:



6.3.9.2.4.3 __dbg_check_enable()

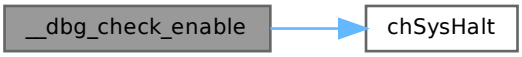
```
void __dbg_check_enable (
              void )
```

Guard code for chSysEnable().

Function Class:

Not an API, this function is for internal use only.

Here is the call graph for this function:



6.3.9.2.4.4 __dbg_check_lock()

```
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:

6.3.9.2.4.5 __dbg_check_unlock()

```
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:

6.3.9.2.4.6 __dbg_check_lock_from_isr()

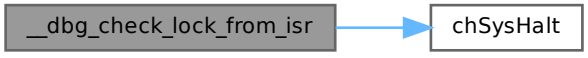
```
void __dbg_check_lock_from_isr (
              void )
```

Guard code for `chSysLockFromIsr()`.

Function Class:

Not an API, this function is for internal use only.

Here is the call graph for this function:



6.3.9.2.4.7 __dbg_check_unlock_from_isr()

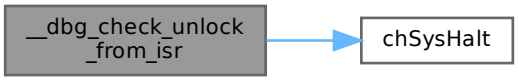
```
void __dbg_check_unlock_from_isr (
              void )
```

Guard code for `chSysUnlockFromIsr()`.

Function Class:

Not an API, this function is for internal use only.

Here is the call graph for this function:



6.3.9.2.4.8 __dbg_check_enter_isr()

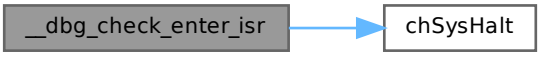
```
void __dbg_check_enter_isr (
              void )
```

Guard code for `CH_IRQ_PROLOGUE()`.

Function Class:

Not an API, this function is for internal use only.

Here is the call graph for this function:



6.3.9.2.4.9 __dbg_check_leave_isr()

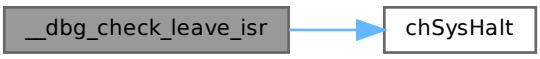
```
void __dbg_check_leave_isr (
              void )
```

Guard code for CH_IRQ_EPILOGUE().

Function Class:

Not an API, this function is for internal use only.

Here is the call graph for this function:



6.3.9.2.4.10 chDbgCheckClassI()

```
void chDbgCheckClassI (
              void )
```

I-class functions context check.

Verifies that the system is in an appropriate state for invoking an I-class API function. A panic is generated if the state is not compatible.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

6.3.9.2.4.11 chDbgCheckClassS()

```
void chDbgCheckClassS (
             void )
```

S-class functions context check.

Verifies that the system is in an appropriate state for invoking an S-class API function. A panic is generated if the state is not compatible.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

6.3.9.2.4.12 __dbg_object_init()

```
void __dbg_object_init (
             system_debug_t * sdp)  [inline], [static]
```

Debug support initialization.

Note

Internal use only.

Parameters

out	*sdp*	pointer to the system_debug_t structure

Function Class:

Not an API, this function is for internal use only.

6.3.9.3 Tracing

6.3.9.3.1 Detailed Description

System events tracing service.

Trace record types

- #define CH_TRACE_TYPE_UNUSED 0U
- #define CH_TRACE_TYPE_READY 1U
- #define CH_TRACE_TYPE_SWITCH 2U
- #define CH_TRACE_TYPE_ISR_ENTER 3U
- #define CH_TRACE_TYPE_ISR_LEAVE 4U
- #define CH_TRACE_TYPE_HALT 5U
- #define CH_TRACE_TYPE_USER 6U

Events to trace

- #define CH_DBG_TRACE_MASK_DISABLED 255U
- #define CH_DBG_TRACE_MASK_NONE 0U
- #define CH_DBG_TRACE_MASK_READY 1U
- #define CH_DBG_TRACE_MASK_SWITCH 2U
- #define CH_DBG_TRACE_MASK_ISR 4U
- #define CH_DBG_TRACE_MASK_HALT 8U
- #define CH_DBG_TRACE_MASK_USER 16U
- #define CH_DBG_TRACE_MASK_SLOW
- #define CH_DBG_TRACE_MASK_ALL

Debug related settings

- #define CH_DBG_TRACE_MASK CH_DBG_TRACE_MASK_DISABLED

  Trace buffer entries.
- #define CH_DBG_TRACE_BUFFER_SIZE 128

  Trace buffer entries.

Data Structures

- struct trace_event_t

  Trace buffer record.
- struct trace_buffer_t

  Trace buffer header.

Macros

- #define __trace_ready(tp, msg)
- #define __trace_switch(ntp, otp)
- #define __trace_isr_enter(isr)
- #define __trace_isr_leave(isr)
- #define __trace_halt(reason)
- #define chDbgWriteTraceI(up1, up2)
- #define chDbgWriteTrace(up1, up2)

Functions

- static NOINLINE void trace_next (os_instance_t *oip)

  Writes a time stamp and increases the trace buffer pointer.
- void __trace_object_init (trace_buffer_t *tbp)

  Circular trace buffer initialization.
- void __trace_ready (thread_t *tp, msg_t msg)

  Inserts in the circular debug trace buffer a ready record.
- void __trace_switch (thread_t *ntp, thread_t *otp)

  Inserts in the circular debug trace buffer a context switch record.
- void __trace_isr_enter (const char *isr)

  Inserts in the circular debug trace buffer an ISR-enter record.
- void __trace_isr_leave (const char *isr)

  Inserts in the circular debug trace buffer an ISR-leave record.
- void __trace_halt (const char *reason)

  Inserts in the circular debug trace buffer an halt record.
- void chTraceWriteI (void *up1, void *up2)

  Adds an user trace record to the trace buffer.
- void chTraceWrite (void *up1, void *up2)

  Adds an user trace record to the trace buffer.
- void chTraceSuspendI (uint16_t mask)

  Suspends one or more trace events.
- void chTraceSuspend (uint16_t mask)

  Suspends one or more trace events.
- void chTraceResumeI (uint16_t mask)

  Resumes one or more trace events.
- void chTraceResume (uint16_t mask)

  Resumes one or more trace events.

6.3.9.3.2 Macro Definition Documentation

6.3.9.3.2.1 CH_TRACE_TYPE_UNUSED

```
#define CH_TRACE_TYPE_UNUSED 0U
```

6.3.9.3.2.2 CH_TRACE_TYPE_READY

```
#define CH_TRACE_TYPE_READY 1U
```

6.3.9.3.2.3 CH_TRACE_TYPE_SWITCH

```
#define CH_TRACE_TYPE_SWITCH 2U
```

6.3.9.3.2.4 CH_TRACE_TYPE_ISR_ENTER

```
#define CH_TRACE_TYPE_ISR_ENTER 3U
```

6.3.9.3.2.5 CH_TRACE_TYPE_ISR_LEAVE

```
#define CH_TRACE_TYPE_ISR_LEAVE 4U
```

6.3.9.3.2.6 CH_TRACE_TYPE_HALT

```
#define CH_TRACE_TYPE_HALT 5U
```

6.3.9.3.2.7 CH_TRACE_TYPE_USER

```
#define CH_TRACE_TYPE_USER 6U
```

6.3.9.3.2.8 CH_DBG_TRACE_MASK_DISABLED

```
#define CH_DBG_TRACE_MASK_DISABLED 255U
```

6.3.9.3.2.9 CH_DBG_TRACE_MASK_NONE

```
#define CH_DBG_TRACE_MASK_NONE 0U
```

6.3.9.3.2.10 CH_DBG_TRACE_MASK_READY

```
#define CH_DBG_TRACE_MASK_READY 1U
```

6.3.9.3.2.11 CH_DBG_TRACE_MASK_SWITCH

```
#define CH_DBG_TRACE_MASK_SWITCH 2U
```

6.3.9.3.2.12 CH_DBG_TRACE_MASK_ISR

```
#define CH_DBG_TRACE_MASK_ISR 4U
```

6.3.9.3.2.13 CH_DBG_TRACE_MASK_HALT

```
#define CH_DBG_TRACE_MASK_HALT 8U
```

6.3.9.3.2.14 CH_DBG_TRACE_MASK_USER

```
#define CH_DBG_TRACE_MASK_USER 16U
```

6.3.9.3.2.15 CH_DBG_TRACE_MASK_SLOW

```
#define CH_DBG_TRACE_MASK_SLOW
```

Value:

```
(CH_DBG_TRACE_MASK_READY |      \
CH_DBG_TRACE_MASK_SWITCH |      \
CH_DBG_TRACE_MASK_HALT |        \
CH_DBG_TRACE_MASK_USER)
```

6.3.9.3.2.16 CH_DBG_TRACE_MASK_ALL

```
#define CH_DBG_TRACE_MASK_ALL
```

Value:

```
(CH_DBG_TRACE_MASK_READY |      \
CH_DBG_TRACE_MASK_SWITCH |      \
CH_DBG_TRACE_MASK_ISR |         \
CH_DBG_TRACE_MASK_HALT |        \
CH_DBG_TRACE_MASK_USER)
```

6.3.9.3.2.17 CH_DBG_TRACE_MASK

```
#define CH_DBG_TRACE_MASK CH_DBG_TRACE_MASK_DISABLED
```

Trace buffer entries.

6.3.9.3.2.18 CH_DBG_TRACE_BUFFER_SIZE

```
#define CH_DBG_TRACE_BUFFER_SIZE 128
```

Trace buffer entries.

Note

The trace buffer is only allocated if CH_DBG_TRACE_MASK is different from CH_DBG_TRACE_MASK_↩ DISABLED.

6.3.9.3.2.19 __trace_ready

```
#define __trace_ready(
              tp,
              msg)
```

6.3.9.3.2.20 __trace_switch

```
#define __trace_switch(
              ntp,
              otp)
```

6.3.9.3.2.21 __trace_isr_enter

```
#define __trace_isr_enter(
              isr)
```

6.3.9.3.2.22 __trace_isr_leave

```
#define __trace_isr_leave(
              isr)
```

6.3.9.3.2.23 __trace_halt

```
#define __trace_halt(
              reason)
```

6.3.9.3.2.24 chDbgWriteTraceI

```
#define chDbgWriteTraceI(
              up1,
              up2)
```

6.3.9.3.2.25 chDbgWriteTrace

```
#define chDbgWriteTrace(
              up1,
              up2)
```

6.3.9.3.3 Function Documentation

6.3.9.3.3.1 trace_next()

```
NOINLINE void trace_next (
             os_instance_t * oip)  [static]
```

Writes a time stamp and increases the trace buffer pointer.

Function Class:

Not an API, this function is for internal use only.

Here is the call graph for this function:

6.3.9.3.3.2 __trace_object_init()

```
void __trace_object_init (
              trace_buffer_t * tbp)
```

Circular trace buffer initialization.

Note

Internal use only.

Parameters

out	*tbp*	pointer to the trace_buffer_t structure

Function Class:

Not an API, this function is for internal use only.

6.3.9.3.3.3 __trace_ready()

```
void __trace_ready (
              thread_t * tp,
              msg_t msg)
```

Inserts in the circular debug trace buffer a ready record.

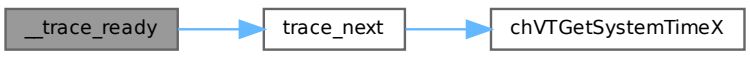
Parameters

in	*tp*	the thread that just become ready
in	*msg*	the thread ready message

Function Class:

Not an API, this function is for internal use only.

Here is the call graph for this function:



6.3.9.3.3.4 __trace_switch()

```
void __trace_switch (
              thread_t * ntp,
              thread_t * otp)
```

Inserts in the circular debug trace buffer a context switch record.

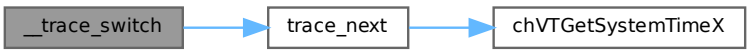
Parameters

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:



6.3.9.3.3.5 __trace_isr_enter()

```
void __trace_isr_enter (
            const char * isr)
```

Inserts in the circular debug trace buffer an ISR-enter record.

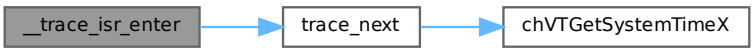
Parameters

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:



6.3.9.3.3.6 __trace_isr_leave()

```
void __trace_isr_leave (
            const char * isr)
```

Inserts in the circular debug trace buffer an ISR-leave record.

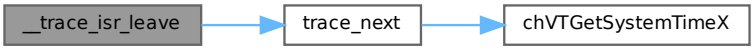
Parameters

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:



6.3.9.3.3.7 __trace_halt()

```
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:

6.3.9.3.3.8 chTraceWriteI()

```
void chTraceWriteI (
            void * up1,
            void * up2)
```

Adds an user trace record to the trace buffer.

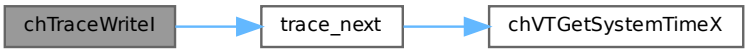
Parameters

in	*up1*	user parameter 1
in	*up2*	user parameter 2

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:



6.3.9.3.3.9 chTraceWrite()

```
void chTraceWrite (
             void * up1,
             void * up2)
```

Adds an user trace record to the trace buffer.

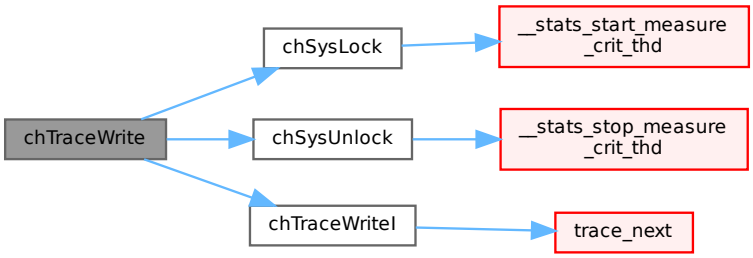
Parameters

in	*up1*	user parameter 1
in	*up2*	user parameter 2

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



6.3.9.3.3.10 chTraceSuspendI()

```
void chTraceSuspendI (
             uint16_t mask)
```

Suspends one or more trace events.

Parameters

in	*mask*	mask of the trace events to be suspended

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

6.3.9.3.3.11 chTraceSuspend()

```
void chTraceSuspend (
             uint16_t mask)
```

Suspends one or more trace events.

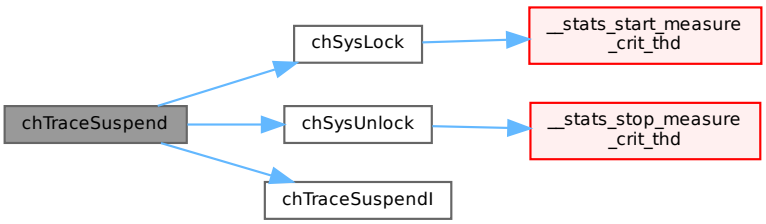
Parameters

in	*mask*	mask of the trace events to be suspended

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



6.3.9.3.3.12 chTraceResumeI()

```
void chTraceResumeI (
             uint16_t mask)
```

Resumes one or more trace events.

Parameters

in	*mask*	mask of the trace events to be resumed

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

6.3.9.3.3.13 chTraceResume()

```
void chTraceResume (
            uint16_t mask)
```

Resumes one or more trace events.

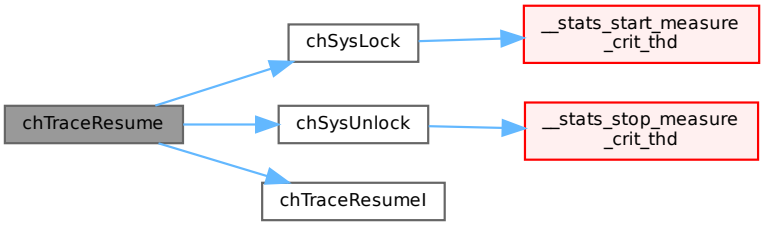
Parameters

in	*mask*	mask of the trace events to be resumed

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



6.3.9.4 Statistics

6.3.9.4.1 Detailed Description

Statistics services.

Data Structures

- struct kernel_stats_t

  Type of a kernel statistics structure.

Functions

- void __stats_increase_irq (void)

  Increases the IRQ counter.
- void __stats_ctxswc (thread_t *ntp, thread_t *otp)

  Updates context switch related statistics.
- void __stats_start_measure_crit_thd (void)

  Starts the measurement of a thread critical zone.
- void __stats_stop_measure_crit_thd (void)

  Stops the measurement of a thread critical zone.
- void __stats_start_measure_crit_isr (void)

  Starts the measurement of an ISR critical zone.
- void __stats_stop_measure_crit_isr (void)

  Stops the measurement of an ISR critical zone.
- void __stats_init (void)
- static void __stats_object_init (kernel_stats_t *ksp)

  Statistics initialization.

6.3.9.4.2 Function Documentation

6.3.9.4.2.1 __stats_increase_irq()

```
void __stats_increase_irq (
              void )
```

Increases the IRQ counter.

6.3.9.4.2.2 __stats_ctxswc()

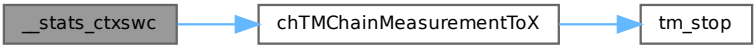
```
void __stats_ctxswc (
              thread_t * ntp,
              thread_t * otp)
```

Updates context switch related statistics.

Parameters

in	*ntp*	the thread to be switched in
in	*otp*	the thread to be switched out

Here is the call graph for this function:

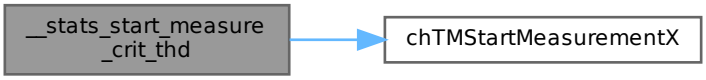


6.3.9.4.2.3 __stats_start_measure_crit_thd()

```
void __stats_start_measure_crit_thd (
            void )
```

Starts the measurement of a thread critical zone.

Here is the call graph for this function:



6.3.9.4.2.4 __stats_stop_measure_crit_thd()

```
void __stats_stop_measure_crit_thd (
            void )
```

Stops the measurement of a thread critical zone.

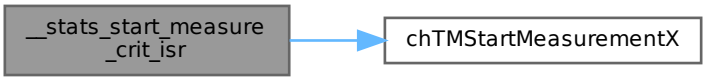
Here is the call graph for this function:

6.3.9.4.2.5 __stats_start_measure_crit_isr()

```
void __stats_start_measure_crit_isr (
            void )
```

Starts the measurement of an ISR critical zone.

Here is the call graph for this function:



6.3.9.4.2.6 __stats_stop_measure_crit_isr()

```
void __stats_stop_measure_crit_isr (
            void )
```

Stops the measurement of an ISR critical zone.

Here is the call graph for this function:

6.3.9.4.2.7 __stats_init()

```
void __stats_init (
            void )
```

6.3.9.4.2.8 __stats_object_init()

```
void __stats_object_init (
            kernel_stats_t * ksp)  [inline], [static]
```

Statistics initialization.

Note

Internal use only.

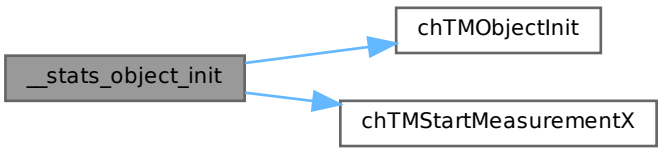
Parameters

out	*ksp*	pointer to the kernel__stats_t structure

Function Class:

Not an API, this function is for internal use only.

Here is the call graph for this function:



6.4 OS Library

6.4.1 Detailed Description

The OS Library is a set of RTOS extensions compatible with both the RT and NIL RTOSes.

Topics

- Version Numbers and Identification
- Synchronization
- Memory Management
- Complex Services

6.4.2 Version Numbers and Identification

6.4.2.1 Detailed Description

OS Library related info.

ChibiOS/LIB version identification

- #define CH_OSLIB_VERSION "1.3.2"

  OS Library version string.
- #define CH_OSLIB_MAJOR 1

  OS Library version major number.
- #define CH_OSLIB_MINOR 3

  OS Library version minor number.
- #define CH_OSLIB_PATCH 2

  OS Library version patch number.

Macros

- #define __CHIBIOS_OSLIB__

  ChibiOS/LIB identification macro.
- #define CH_OSLIB_STABLE 1

  Stable release flag.
- #define CH_CFG_USE_FACTORY FALSE
- #define CH_CFG_USE_HEAP FALSE
- #define CH_CFG_USE_MEMPOOLS FALSE
- #define CH_CFG_USE_OBJ_FIFOS FALSE
- #define CH_CFG_USE_PIPES FALSE
- #define CH_CFG_USE_OBJ_CACHES FALSE
- #define CH_CFG_USE_DELEGATES FALSE
- #define CH_CFG_USE_JOBS FALSE
- #define CH_CFG_USE_MAILBOXES FALSE

Functions

- static void __oslib_init (void)

  Initialization of all library modules.

6.4.2.2 Macro Definition Documentation

6.4.2.2.1 __CHIBIOS_OSLIB__

```
#define __CHIBIOS_OSLIB__
```

ChibiOS/LIB identification macro.

6.4.2.2.2 CH_OSLIB_STABLE

```
#define CH_OSLIB_STABLE 1
```

Stable release flag.

6.4.2.2.3 CH_OSLIB_VERSION

```
#define CH_OSLIB_VERSION "1.3.2"
```

OS Library version string.

6.4.2.2.4 CH_OSLIB_MAJOR

```
#define CH_OSLIB_MAJOR 1
```

OS Library version major number.

6.4.2.2.5 CH_OSLIB_MINOR

```
#define CH_OSLIB_MINOR 3
```

OS Library version minor number.

6.4.2.2.6 CH_OSLIB_PATCH

```
#define CH_OSLIB_PATCH 2
```

OS Library version patch number.

6.4.2.2.7 CH_CFG_USE_FACTORY

```
#define CH_CFG_USE_FACTORY FALSE
```

6.4.2.2.8 CH_CFG_USE_HEAP

```
#define CH_CFG_USE_HEAP FALSE
```

6.4.2.2.9 CH_CFG_USE_MEMPOOLS

```
#define CH_CFG_USE_MEMPOOLS FALSE
```

6.4.2.2.10 CH_CFG_USE_OBJ_FIFOS

```
#define CH_CFG_USE_OBJ_FIFOS FALSE
```

6.4.2.2.11 CH_CFG_USE_PIPES

```
#define CH_CFG_USE_PIPES FALSE
```

6.4.2.2.12 CH_CFG_USE_OBJ_CACHES

```
#define CH_CFG_USE_OBJ_CACHES FALSE
```

6.4.2.2.13 CH_CFG_USE_DELEGATES

```
#define CH_CFG_USE_DELEGATES FALSE
```

6.4.2.2.14 CH_CFG_USE_JOBS

```
#define CH_CFG_USE_JOBS FALSE
```

6.4.2.2.15 CH_CFG_USE_MAILBOXES

```
#define CH_CFG_USE_MAILBOXES FALSE
```

6.4.2.3 Function Documentation

6.4.2.3.1 __oslib_init()

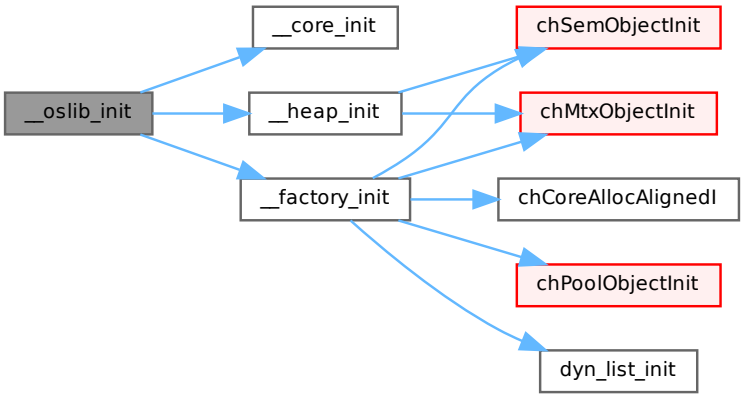
```
void __oslib_init (
             void )  [inline], [static]
```

Initialization of all library modules.

Function Class:

Not an API, this function is for internal use only.

Here is the call graph for this function:



6.4.3 Synchronization

6.4.3.1 Detailed Description

Synchronization services.

Topics

- Binary Semaphores
- Mailboxes
- Pipes
- Delegate Threads
- Jobs Queues

6.4.3.2 Binary Semaphores

6.4.3.2.1 Detailed Description

Data Structures

- struct ch_binary_semaphore

  Binary semaphore type.

Macros

- #define __BSEMAPHORE_DATA(name, taken)

  Data part of a static semaphore initializer.
- #define BSEMAPHORE_DECL(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.

6.4.3.2.2 Macro Definition Documentation

6.4.3.2.2.1 __BSEMAPHORE_DATA

```
#define __BSEMAPHORE_DATA(
              name,
              taken)
```

Value:

```
{__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

in	*name*	the name of the semaphore variable
in	*taken*	the semaphore initial state

6.4.3.2.2.2 BSEMAPHORE_DECL

```
#define BSEMAPHORE_DECL(
              name,
              taken)
```

Value:

```
binary_semaphore_t name = __BSEMAPHORE_DATA(name, taken)
```

Static semaphore initializer.

Statically initialized semaphores require no explicit initialization using `chBSemInit()`.

Parameters

in	*name*	the name of the semaphore variable
in	*taken*	the semaphore initial state

6.4.3.2.3 Typedef Documentation

6.4.3.2.3.1 binary_semaphore_t

```
typedef struct ch_binary_semaphore binary_semaphore_t
```

Binary semaphore type.

6.4.3.2.4 Function Documentation

6.4.3.2.4.1 chBSemObjectInit()

```
void chBSemObjectInit (
             binary_semaphore_t * bsp,
             bool taken)  [inline], [static]
```

Initializes a binary semaphore.

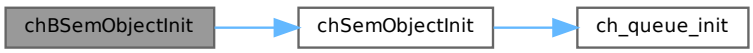
Parameters

out	*bsp*	pointer to a `binary_semaphore_t` structure
in	*taken*	initial state of the binary semaphore: • *false*, the initial state is not taken. • *true*, the initial state is taken.

Function Class:

Object or module nitializer function.

Here is the call graph for this function:



6.4.3.2.4.2 chBSemWait()

```
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.

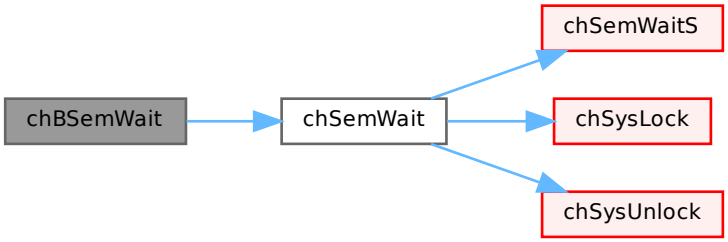
Return values

MSG_OK	if the binary semaphore has been successfully taken.
MSG_RESET	if the binary semaphore has been reset using `chBSemReset()`.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



6.4.3.2.4.3 chBSemWaitS()

```
msg_t chBSemWaitS (
            binary_semaphore_t * bsp)  [inline], [static]
```

Wait operation on the binary semaphore.

Parameters

in	*bsp*	pointer to a `binary_semaphore_t` structure

Returns

A message specifying how the invoking thread has been released from the semaphore.

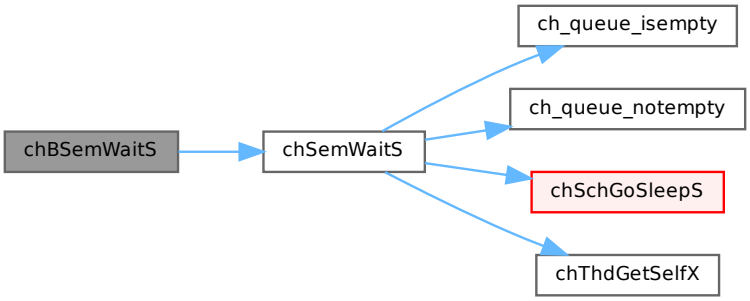
Return values

MSG_OK	if the binary semaphore has been successfully taken.
MSG_RESET	if the binary semaphore has been reset using `chBSemReset()`.

Function Class:

This is an **S-Class** API, this function can be invoked from within a system lock zone by threads only.

Here is the call graph for this function:



6.4.3.2.4.4 chBSemWaitTimeoutS()

```
msg_t chBSemWaitTimeoutS (
            binary_semaphore_t * bsp,
            sysinterval_t timeout)  [inline], [static]
```

Wait operation on the binary semaphore.

Parameters

in	*bsp*	pointer to a `binary_semaphore_t` structure
in	*timeout*	the number of ticks before the operation timeouts, the following special values are allowed: • *TIME_IMMEDIATE* immediate timeout. • *TIME_INFINITE* no timeout.

Returns

A message specifying how the invoking thread has been released from the semaphore.

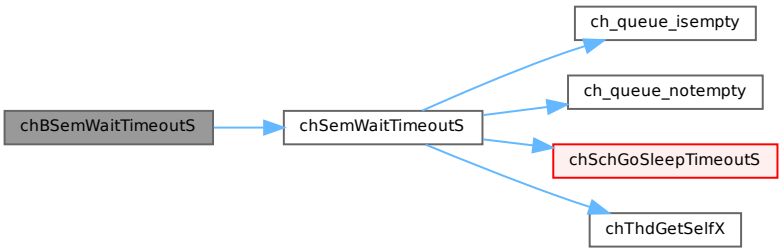
Return values

MSG_OK	if the binary semaphore has been successfully taken.
MSG_RESET	if the binary semaphore has been reset using `chBSemReset()`.
MSG_TIMEOUT	if the binary semaphore has not been signaled or reset within the specified timeout.

Function Class:

This is an **S-Class** API, this function can be invoked from within a system lock zone by threads only.

Here is the call graph for this function:



6.4.3.2.4.5 chBSemWaitTimeout()

```
msg_t chBSemWaitTimeout (
            binary_semaphore_t * bsp,
            sysinterval_t timeout)  [inline], [static]
```

Wait operation on the binary semaphore.

Parameters

in	*bsp*	pointer to a `binary_semaphore_t` structure

in	*timeout*	the number of ticks before the operation timeouts, the following special values are allowed: • *TIME_IMMEDIATE* immediate timeout. • *TIME_INFINITE* no timeout.

Returns

A message specifying how the invoking thread has been released from the semaphore.

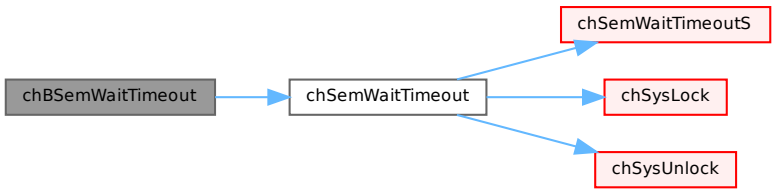
Return values

MSG_OK	if the binary semaphore has been successfully taken.
MSG_RESET	if the binary semaphore has been reset using `chBSemReset()`.
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:



6.4.3.2.4.6 chBSemResetI()

```
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 `chBSemWait()` will return `MSG_RESET` instead of `MSG_OK`.

This function does not reschedule.

Parameters

in	*bsp*	pointer to a `binary_semaphore_t` structure
in	*taken*	new state of the binary semaphore • *false*, the new state is not taken. • *true*, the new state is taken.

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:

6.4.3.2.4.7 chBSemReset()

```
void chBSemReset (
            binary_semaphore_t * bsp,
            bool taken)  [inline], [static]
```

Reset operation on the binary semaphore.

Note

The released threads can recognize they were waked up by a reset rather than a signal because the `chBSemWait()` will return `MSG_RESET` instead of `MSG_OK`.

Parameters

in	*bsp*	pointer to a `binary_semaphore_t` structure
in	*taken*	new state of the binary semaphore • *false*, the new state is not taken. • *true*, the new state is taken.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

6.4.3.2.4.8 chBSemSignalI()

```
void chBSemSignalI (
            binary_semaphore_t * bsp)  [inline], [static]
```

Performs a signal operation on a binary semaphore.

Note

This function does not reschedule.

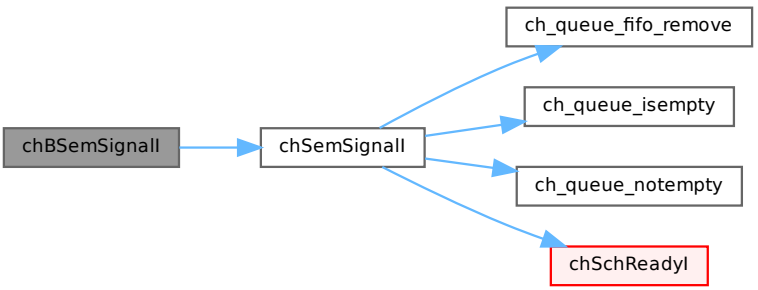
Parameters

in	*bsp*	pointer to a `binary_semaphore_t` structure

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:



6.4.3.2.4.9 chBSemSignal()

```
void chBSemSignal (
            binary_semaphore_t * bsp)  [inline], [static]
```

Performs a signal operation on a binary semaphore.

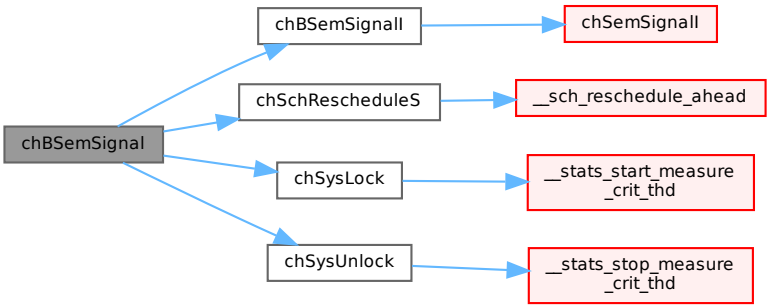
Parameters

in	*bsp*	pointer to a binary_semaphore_t structure

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



6.4.3.2.4.10 chBSemGetStateI()

```
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.

6.4.3.3 Mailboxes

6.4.3.3.1 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.

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)

  Static mailbox initializer.

Functions

- void chMBObjectInit (mailbox_t *mbp, msg_t *buf, size_t n)

  Initializes a `mailbox_t` object.
- void chMBReset (mailbox_t *mbp)

  Resets a `mailbox_t` object.
- void chMBResetI (mailbox_t *mbp)

  Resets a `mailbox_t` object.
- msg_t chMBPostTimeout (mailbox_t *mbp, msg_t msg, sysinterval_t timeout)

  Posts a message into a mailbox.
- msg_t chMBPostTimeoutS (mailbox_t *mbp, msg_t msg, sysinterval_t timeout)

  Posts a message into a mailbox.
- msg_t chMBPostI (mailbox_t *mbp, msg_t msg)

  Posts a message into a mailbox.
- msg_t chMBPostAheadTimeout (mailbox_t *mbp, msg_t msg, sysinterval_t timeout)

  Posts an high priority message into a mailbox.
- msg_t chMBPostAheadTimeoutS (mailbox_t *mbp, msg_t msg, sysinterval_t timeout)

  Posts an high priority message into a mailbox.
- msg_t chMBPostAheadI (mailbox_t *mbp, msg_t msg)

  Posts an high priority message into a mailbox.
- msg_t chMBFetchTimeout (mailbox_t *mbp, msg_t *msgp, sysinterval_t timeout)

  Retrieves a message from a mailbox.
- msg_t chMBFetchTimeoutS (mailbox_t *mbp, msg_t *msgp, sysinterval_t timeout)

  Retrieves a message from a mailbox.
- msg_t chMBFetchI (mailbox_t *mbp, msg_t *msgp)

  Retrieves a message from a mailbox.
- static size_t chMBGetSizeI (const mailbox_t *mbp)

  Returns the mailbox buffer size as number of messages.
- static size_t chMBGetUsedCountI (const mailbox_t *mbp)

  Returns the number of used message slots into a mailbox.
- static size_t chMBGetFreeCountI (const mailbox_t *mbp)

  Returns the number of free message slots into a mailbox.
- static msg_t chMBPeekI (const mailbox_t *mbp)

  Returns the next message in the queue without removing it.
- static void chMBResumeX (mailbox_t *mbp)

  Terminates the reset state.

6.4.3.3.2 Macro Definition Documentation

6.4.3.3.2.1 __MAILBOX_DATA

```
#define __MAILBOX_DATA(
              name,
              buffer,
              size)
```

Value:

```
  {                                 \
  (msg_t *)(buffer),                                    \
  (msg_t *)(buffer) + size,                             \
  (msg_t *)(buffer),                                    \
```

```
  (msg_t *)(buffer),                                                    \
  (size_t)0,                                                            \
  false,                                                                \
  __THREADS_QUEUE_DATA(name.qw),                                        \
  __THREADS_QUEUE_DATA(name.qr),                                        \
}
```

Data part of a static mailbox initializer.

This macro should be used when statically initializing a mailbox that is part of a bigger structure.

Parameters

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

6.4.3.3.2.2 MAILBOX_DECL

```
#define MAILBOX_DECL(
              name,
              buffer,
              size)
```

Value:

```
  mailbox_t name = __MAILBOX_DATA(name, buffer, size)
```

Static mailbox initializer.

Statically initialized mailboxes require no explicit initialization using chMBObjectInit().

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

6.4.3.3.3 Function Documentation

6.4.3.3.3.1 chMBObjectInit()

```
void chMBObjectInit (
             mailbox_t * mbp,
             msg_t * buf,
             size_t n)
```

Initializes a mailbox_t object.

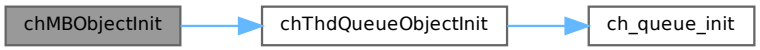
Parameters

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:

Object or module nitializer function.

Here is the call graph for this function:



6.4.3.3.3.2 chMBReset()

```
void chMBReset (
              mailbox_t * mbp)
```

Resets a `mailbox_t` object.

All the waiting threads are resumed with status `MSG_RESET` and the queued messages are lost.

Postcondition

The mailbox is in reset state, all operations will fail and return `MSG_RESET` until the mailbox is enabled again using `chMBResumeX()`.

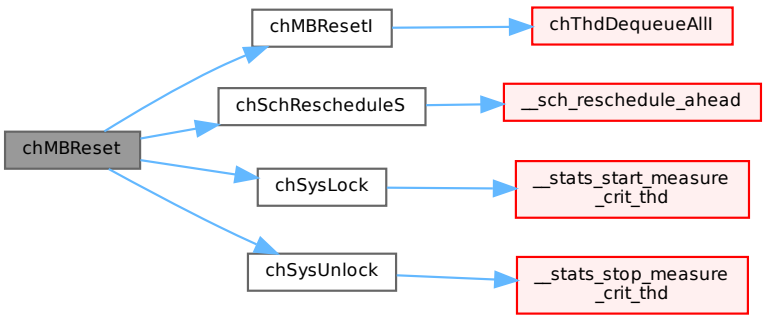
Parameters

`in`	*mbp*	the pointer to an initialized `mailbox_t` object

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



6.4.3.3.3.3 chMBResetI()

```
void chMBResetI (
             mailbox_t * mbp)
```

Resets a mailbox_t object.

All the waiting threads are resumed with status MSG_RESET and the queued messages are lost.

Postcondition

The mailbox is in reset state, all operations will fail and return MSG_RESET until the mailbox is enabled again using chMBResumeX().

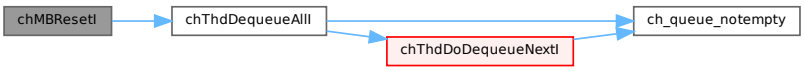
Parameters

in	*mbp*	the pointer to an initialized mailbox_t object

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



6.4.3.3.3.4 chMBPostTimeout()

```
msg_t chMBPostTimeout (
             mailbox_t * mbp,
             msg_t msg,
             sysinterval_t timeout)
```

Posts a message into a mailbox.

The invoking thread waits until a empty slot in the mailbox becomes available or the specified time runs out.

Parameters

in	*mbp*	the pointer to an initialized mailbox_t object
in	*msg*	the message to be posted on the mailbox
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.

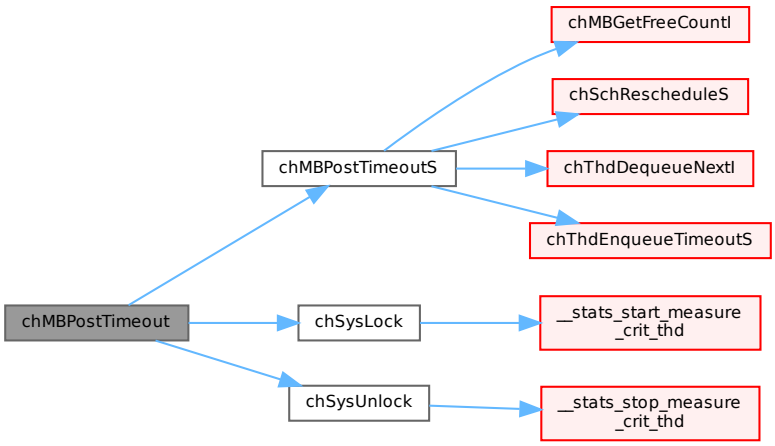
Return values

MSG_OK	if a message has been correctly posted.
MSG_RESET	if the mailbox has been reset.
MSG_TIMEOUT	if the operation has timed out.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



6.4.3.3.3.5 chMBPostTimeoutS()

```
msg_t chMBPostTimeoutS (
            mailbox_t * mbp,
            msg_t msg,
            sysinterval_t timeout)
```

Posts a message into a mailbox.

The invoking thread waits until a empty slot in the mailbox becomes available or the specified time runs out.

Parameters

in	*mbp*	the pointer to an initialized `mailbox_t` object
in	*msg*	the message to be posted on the mailbox
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.

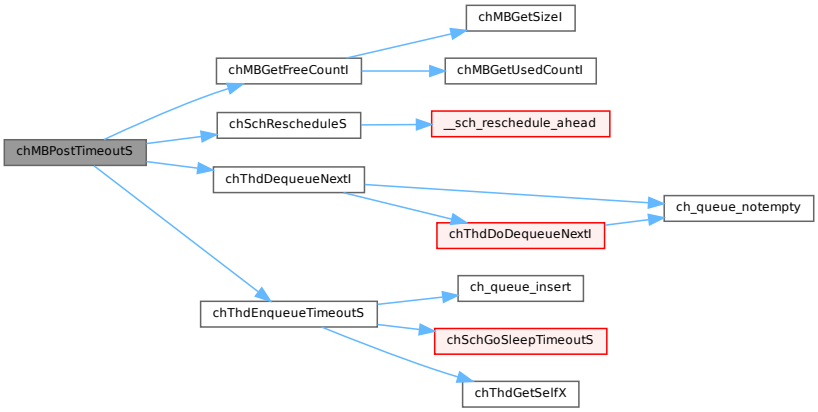
Return values

MSG_OK	if a message has been correctly posted.
MSG_RESET	if the mailbox has been reset.
MSG_TIMEOUT	if the operation has timed out.

Function Class:

This is an **S-Class** API, this function can be invoked from within a system lock zone by threads only.

Here is the call graph for this function:



6.4.3.3.3.6 chMBPostI()

```
msg_t chMBPostI (
            mailbox_t * mbp,
            msg_t msg)
```

Posts a message into a mailbox.

This variant is non-blocking, the function returns a timeout condition if the queue is full.

Parameters

in	*mbp*	the pointer to an initialized `mailbox_t` object
in	*msg*	the message to be posted on the mailbox

Returns

The operation status.

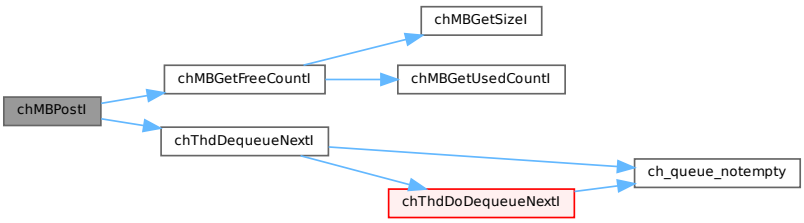
Return values

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:



6.4.3.3.3.7 chMBPostAheadTimeout()

```
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

in	*mbp*	the pointer to an initialized `mailbox_t` object
in	*msg*	the message to be posted on the mailbox
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.

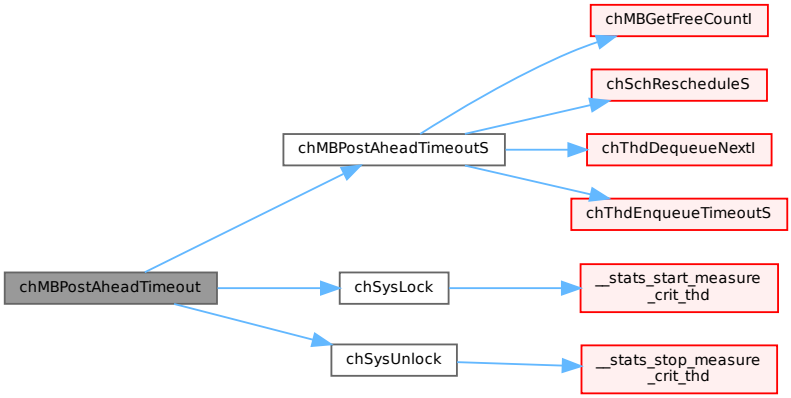
Return values

MSG_OK	if a message has been correctly posted.
MSG_RESET	if the mailbox has been reset.
MSG_TIMEOUT	if the operation has timed out.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



6.4.3.3.3.8 chMBPostAheadTimeoutS()

```
msg_t chMBPostAheadTimeoutS (
            mailbox_t * mbp,
            msg_t msg,
            sysinterval_t timeout)
```

Posts an high priority message into a mailbox.

The invoking thread waits until a empty slot in the mailbox becomes available or the specified time runs out.

Parameters

in	*mbp*	the pointer to an initialized `mailbox_t` object
in	*msg*	the message to be posted on the mailbox
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.

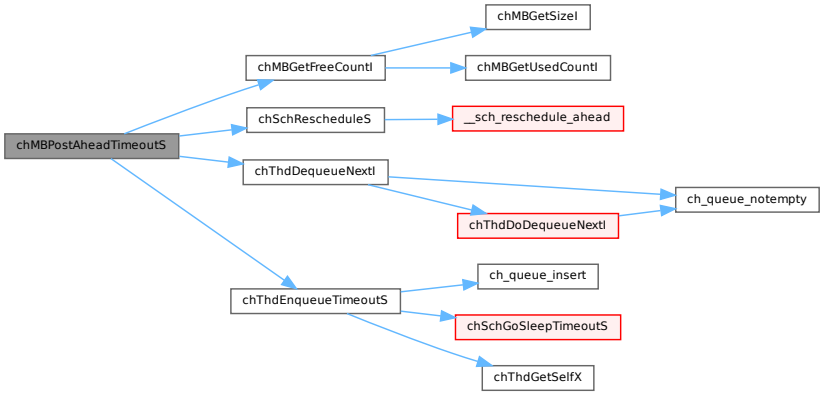
Return values

MSG_OK	if a message has been correctly posted.
MSG_RESET	if the mailbox has been reset.
MSG_TIMEOUT	if the operation has timed out.

Function Class:

This is an **S-Class** API, this function can be invoked from within a system lock zone by threads only.

Here is the call graph for this function:



6.4.3.3.3.9 chMBPostAheadI()

```
msg_t chMBPostAheadI (
              mailbox_t * mbp,
              msg_t msg)
```

Posts an high priority message into a mailbox.

This variant is non-blocking, the function returns a timeout condition if the queue is full.

Parameters

in	*mbp*	the pointer to an initialized `mailbox_t` object
in	*msg*	the message to be posted on the mailbox

Returns

The operation status.

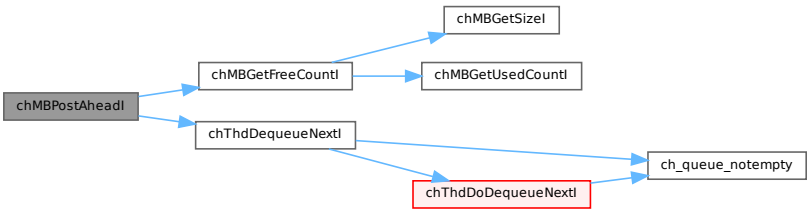
Return values

MSG_OK	if a message has been correctly posted.
MSG_RESET	if the mailbox has been reset.
MSG_TIMEOUT	if the mailbox is full and the message cannot be posted.

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:



6.4.3.3.3.10 chMBFetchTimeout()

```
msg_t chMBFetchTimeout (
            mailbox_t * mbp,
            msg_t * msgp,
            sysinterval_t timeout)
```

Retrieves a message from a mailbox.

The invoking thread waits until a message is posted in the mailbox or the specified time runs out.

Parameters

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.

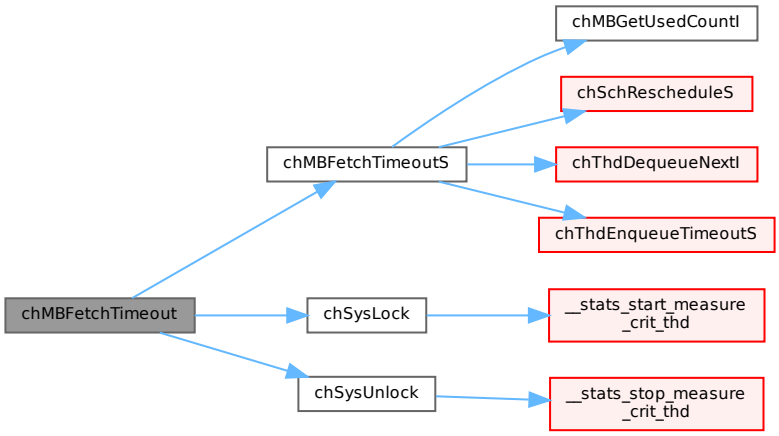
Return values

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:



6.4.3.3.3.11 chMBFetchTimeoutS()

```
msg_t chMBFetchTimeoutS (
            mailbox_t * mbp,
            msg_t * msgp,
            sysinterval_t timeout)
```

Retrieves a message from a mailbox.

The invoking thread waits until a message is posted in the mailbox or the specified time runs out.

Parameters

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.

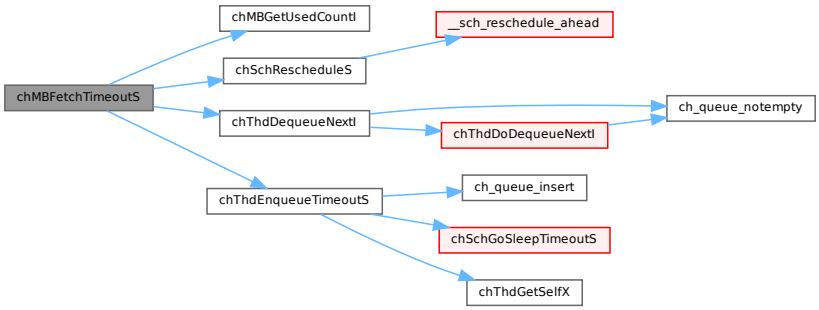
Return values

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:

This is an **S-Class** API, this function can be invoked from within a system lock zone by threads only.

Here is the call graph for this function:



6.4.3.3.3.12 chMBFetchI()

```
msg_t chMBFetchI (
              mailbox_t * mbp,
              msg_t * msgp)
```

Retrieves a message from a mailbox.

This variant is non-blocking, the function returns a timeout condition if the queue is empty.

Parameters

in	*mbp*	the pointer to an initialized `mailbox_t` object
out	*msgp*	pointer to a message variable for the received message

Returns

The operation status.

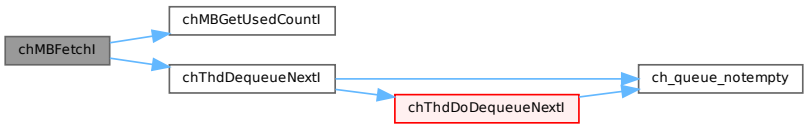
Return values

MSG_OK	if a message has been correctly fetched.
MSG_RESET	if the mailbox has been reset.
MSG_TIMEOUT	if the mailbox is empty and a message cannot be fetched.

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:



6.4.3.3.3.13 chMBGetSizeI()

```
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.

6.4.3.3.3.14 chMBGetUsedCountI()

```
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.

6.4.3.3.3.15 chMBGetFreeCountI()

```
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

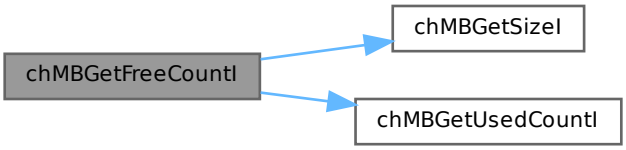
Returns

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:



6.4.3.3.3.16 chMBPeekI()

```
msg_t chMBPeekI (
            const mailbox_t * mbp)  [inline], [static]
```

Returns the next message in the queue without removing it.

Precondition

A message must be waiting in the queue for this function to work or it would return garbage. The correct way to use this macro is to use chMBGetUsedCountI() and then use this macro, all within a lock state.

Parameters

in	*mbp*	the pointer to an initialized mailbox_t object

Returns

The next message in queue.

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

6.4.3.3.3.17 chMBResumeX()

```
void chMBResumeX (
            mailbox_t * mbp)  [inline], [static]
```

Terminates the reset state.

Parameters

in	*mbp*	the pointer to an initialized mailbox_t object

Function Class:

This is an **X-Class** API, this function can be invoked from any context.

6.4.3.4 Pipes

6.4.3.4.1 Detailed Description

Data Structures

- struct pipe_t

  Structure representing a pipe object.

Macros

- #define PC_INIT(p)
- #define PC_LOCK(p)
- #define PC_UNLOCK(p)
- #define PW_INIT(p)
- #define PW_LOCK(p)
- #define PW_UNLOCK(p)
- #define PR_INIT(p)
- #define PR_LOCK(p)
- #define PR_UNLOCK(p)
- #define __PIPE_DATA(name, buffer, size)

  Data part of a static pipe initializer.
- #define PIPE_DECL(name, buffer, size)

  Static pipe initializer.

Functions

- static size_t pipe_write (pipe_t *pp, const uint8_t *bp, size_t n)

  Non-blocking pipe write.
- static size_t pipe_read (pipe_t *pp, uint8_t *bp, size_t n)

  Non-blocking pipe read.
- void chPipeObjectInit (pipe_t *pp, uint8_t *buf, size_t n)

  Initializes a `mailbox_t` object.
- void chPipeReset (pipe_t *pp)

  Resets a `pipe_t` object.
- size_t chPipeWriteTimeout (pipe_t *pp, const uint8_t *bp, size_t n, sysinterval_t timeout)

  Pipe write with timeout.
- size_t chPipeReadTimeout (pipe_t *pp, uint8_t *bp, size_t n, sysinterval_t timeout)

  Pipe read with timeout.
- static size_t chPipeGetSize (const pipe_t *pp)

  Returns the pipe buffer size as number of bytes.
- static size_t chPipeGetUsedCount (const pipe_t *pp)

  Returns the number of used byte slots into a pipe.
- static size_t chPipeGetFreeCount (const pipe_t *pp)

  Returns the number of free byte slots into a pipe.
- static void chPipeResume (pipe_t *pp)

  Terminates the reset state.

6.4.3.4.2 Macro Definition Documentation

6.4.3.4.2.1 PC_INIT

```
#define PC_INIT(
                p)
```

Value:

```
chMtxObjectInit(&(p)->cmtx)
```

6.4.3.4.2.2 PC_LOCK

```
#define PC_LOCK(
                p)
```

Value:

```
chMtxLock(&(p)->cmtx)
```

6.4.3.4.2.3 PC_UNLOCK

```
#define PC_UNLOCK(
                p)
```

Value:

```
chMtxUnlock(&(p)->cmtx)
```

6.4.3.4.2.4 PW_INIT

```
#define PW_INIT(
                p)
```

Value:

```
chMtxObjectInit(&(p)->wmtx)
```

6.4.3.4.2.5 PW_LOCK

```
#define PW_LOCK(
                p)
```

Value:

```
chMtxLock(&(p)->wmtx)
```

6.4.3.4.2.6 PW_UNLOCK

```
#define PW_UNLOCK(
                p)
```

Value:

```
chMtxUnlock(&(p)->wmtx)
```

6.4.3.4.2.7 PR_INIT

```
#define PR_INIT(
                p)
```

Value:

```
chMtxObjectInit(&(p)->rmtx)
```

6.4.3.4.2.8 PR_LOCK

```
#define PR_LOCK(
                p)
```

Value:

```
chMtxLock(&(p)->rmtx)
```

6.4.3.4.2.9 PR_UNLOCK

```
#define PR_UNLOCK(
                p)
```

Value:

```
chMtxUnlock(&(p)->rmtx)
```

6.4.3.4.2.10 __PIPE_DATA

```
#define __PIPE_DATA(
              name,
              buffer,
              size)
```

Value:

```
  {                                   \
  (uint8_t *)(buffer),                                    \
  (uint8_t *)(buffer) + size,                             \
  (uint8_t *)(buffer),                                    \
  (uint8_t *)(buffer),                                    \
  (size_t)0,                                              \
  false,                                                  \
  NULL,                                                   \
  NULL,                                                   \
  __MUTEX_DATA(name.cmtx),                                \
  __MUTEX_DATA(name.wmtx),                                \
  __MUTEX_DATA(name.rmtx),                                \
}
```

Data part of a static pipe initializer.

This macro should be used when statically initializing a pipe that is part of a bigger structure.

Parameters

in	*name*	the name of the pipe variable
in	*buffer*	pointer to the pipe buffer array of `uint8_t`
in	*size*	number of `uint8_t` elements in the buffer array

6.4.3.4.2.11 PIPE_DECL

```
#define PIPE_DECL(
              name,
              buffer,
              size)
```

Value:

```
  pipe_t name = __PIPE_DATA(name, buffer, size)
```

Static pipe initializer.

Statically initialized pipes require no explicit initialization using `chPipeObjectInit()`.

Parameters

in	*name*	the name of the pipe variable
in	*buffer*	pointer to the pipe buffer array of `uint8_t`
in	*size*	number of `uint8_t` elements in the buffer array

6.4.3.4.3 Function Documentation

6.4.3.4.3.1 pipe_write()

```
size_t pipe_write (
            pipe_t * pp,
            const uint8_t * bp,
            size_t n)  [static]
```

Non-blocking pipe write.

The function writes data from a buffer to a pipe. The operation completes when the specified amount of data has been transferred or when the pipe buffer has been filled.

Parameters

in	*pp*	the pointer to an initialized `pipe_t` object
in	*bp*	pointer to the data buffer
in	*n*	the maximum amount of data to be transferred, the value 0 is reserved

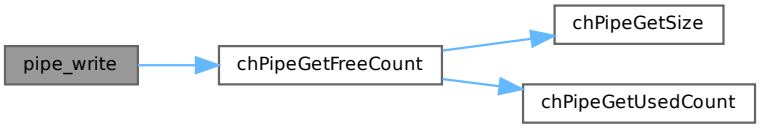
Returns

The number of bytes effectively transferred.

Function Class:

Not an API, this function is for internal use only.

Here is the call graph for this function:



6.4.3.4.3.2 pipe_read()

```
size_t pipe_read (
            pipe_t * pp,
            uint8_t * bp,
            size_t n)  [static]
```

Non-blocking pipe read.

The function reads data from a pipe into a buffer. The operation completes when the specified amount of data has been transferred or when the pipe buffer has been emptied.

Parameters

in	*pp*	the pointer to an initialized pipe_t object
out	*bp*	pointer to the data buffer
in	*n*	the maximum amount of data to be transferred, the value 0 is reserved

Returns

The number of bytes effectively transferred.

Function Class:

Not an API, this function is for internal use only.

Here is the call graph for this function:

6.4.3.4.3.3 chPipeObjectInit()

```
void chPipeObjectInit (
              pipe_t * pp,
              uint8_t * buf,
              size_t n)
```

Initializes a mailbox_t object.

Parameters

out	*pp*	the pointer to the pipe_t structure to be initialized
in	*buf*	pointer to the pipe buffer as an array of uint8_t
in	*n*	number of elements in the buffer array

Function Class:

Object or module nitializer function.

6.4.3.4.3.4 chPipeReset()

```
void chPipeReset (
              pipe_t * pp)
```

Resets a pipe_t object.

All the waiting threads are resumed with status MSG_RESET and the queued data is lost.

Postcondition

The pipe is in reset state, all operations will fail and return MSG_RESET until the mailbox is enabled again using chPipeResumeX().

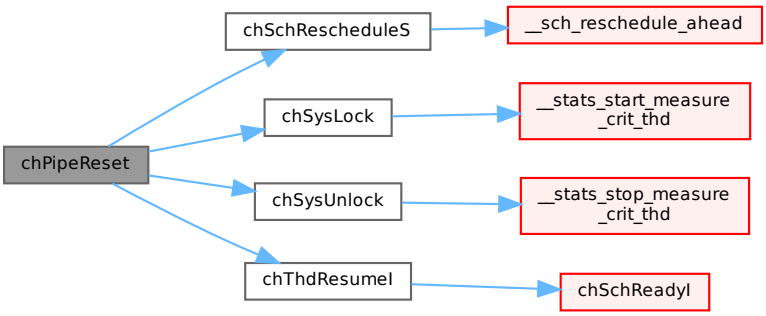
Parameters

in	*pp*	the pointer to an initialized pipe_t object

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



6.4.3.4.3.5 chPipeWriteTimeout()

```
size_t chPipeWriteTimeout (
              pipe_t * pp,
              const uint8_t * bp,
              size_t n,
              sysinterval_t timeout)
```

Pipe write with timeout.

The function writes data from a buffer to a pipe. The operation completes when the specified amount of data has been transferred or after the specified timeout or if the pipe has been reset.

Parameters

in	*pp*	the pointer to an initialized pipe_t object
in	*bp*	pointer to the data buffer
in	*n*	the number of bytes to be written, the value 0 is reserved
in	*timeout*	the number of ticks before the operation timeouts, the following special values are allowed: • *TIME_IMMEDIATE* immediate timeout. • *TIME_INFINITE* no timeout.

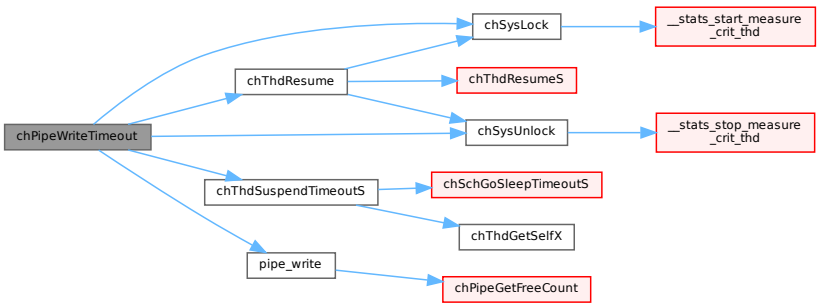
Returns

The number of bytes effectively transferred. A number lower than `n` means that a timeout occurred or the pipe went in reset state.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



6.4.3.4.3.6 chPipeReadTimeout()

```
size_t chPipeReadTimeout (
            pipe_t * pp,
            uint8_t * bp,
            size_t n,
            sysinterval_t timeout)
```

Pipe read with timeout.

The function reads data from a pipe into a buffer. The operation completes when the specified amount of data has been transferred or after the specified timeout or if the pipe has been reset.

Parameters

`in`	*pp*	the pointer to an initialized `pipe_t` object
`out`	*bp*	pointer to the data buffer
`in`	*n*	the number of bytes to be read, the value 0 is reserved
`in`	*timeout*	the number of ticks before the operation timeouts, the following special values are allowed: • *TIME_IMMEDIATE* immediate timeout. • *TIME_INFINITE* no timeout.

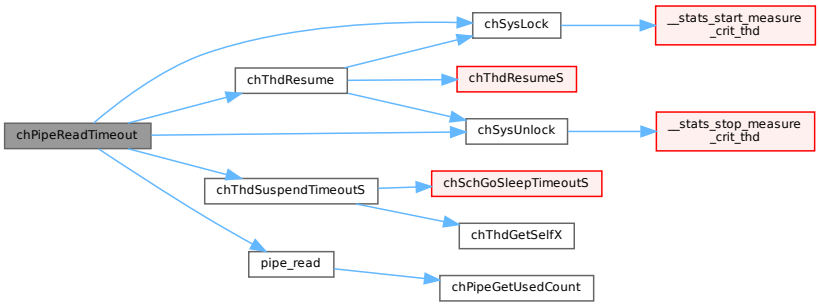
Returns

The number of bytes effectively transferred. A number lower than `n` means that a timeout occurred or the pipe went in reset state.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



6.4.3.4.3.7 chPipeGetSize()

```
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.

6.4.3.4.3.8 chPipeGetUsedCount()

```
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.

6.4.3.4.3.9 chPipeGetFreeCount()

```
size_t chPipeGetFreeCount (
             const pipe_t * pp)  [inline], [static]
```

Returns the number of free byte slots into a pipe.

Parameters

in	*pp*	the pointer to an initialized pipe_t object

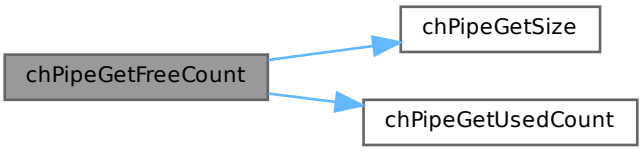
Returns

The number of empty byte slots.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



6.4.3.4.3.10 chPipeResume()

```
void chPipeResume (
             pipe_t * pp)  [inline], [static]
```

Terminates the reset state.

Parameters

in	*pp*	the pointer to an initialized pipe_t object

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

6.4.3.5 Delegate Threads

6.4.3.5.1 Detailed Description

Typedefs

- typedef msg_t(∗ delegate_veneer_t) (va_list ∗argsp)

  Type of a delegate veneer function.
- typedef msg_t(∗ delegate_fn0_t) (void)

  Type of a delegate function with no parameters.
- typedef msg_t(∗ delegate_fn1_t) (msg_t p1)

  Type of a delegate function with one parameter.
- typedef msg_t(∗ delegate_fn2_t) (msg_t p1, msg_t p2)

  Type of a delegate function with two parameters.
- typedef msg_t(∗ delegate_fn3_t) (msg_t p1, msg_t p2, msg_t p3)

  Type of a delegate function with three parameters.
- typedef msg_t(∗ delegate_fn4_t) (msg_t p1, msg_t p2, msg_t p3, msg_t p4)

  Type of a delegate function with four parameters.

Functions

- msg_t __ch_delegate_fn0 (va_list ∗argsp)

  Veneer for functions with no parameters.
- msg_t __ch_delegate_fn1 (va_list ∗argsp)

  Veneer for functions with one parameter.
- msg_t __ch_delegate_fn2 (va_list ∗argsp)

  Veneer for functions with two parameters.
- msg_t __ch_delegate_fn3 (va_list ∗argsp)

  Veneer for functions with three parameters.
- msg_t __ch_delegate_fn4 (va_list ∗argsp)

  Veneer for functions with four parameters.
- msg_t chDelegateCallVeneer (thread_t ∗tp, delegate_veneer_t veneer,...)

  Triggers a function call on a delegate thread.
- void chDelegateDispatch (void)

  Call messages dispatching.
- msg_t chDelegateDispatchTimeout (sysinterval_t timeout)

  Call messages dispatching with timeout.
- static msg_t chDelegateCallDirect0 (thread_t ∗tp, delegate_fn0_t func)

  Direct call to a function with no parameters.
- static msg_t chDelegateCallDirect1 (thread_t ∗tp, delegate_fn1_t func, msg_t p1)

  Direct call to a function with one parameter.
- static msg_t chDelegateCallDirect2 (thread_t ∗tp, delegate_fn2_t func, msg_t p1, msg_t p2)

  Direct call to a function with two parameters.
- static msg_t chDelegateCallDirect3 (thread_t ∗tp, delegate_fn3_t func, msg_t p1, msg_t p2, msg_t p3)

  Direct call to a function with three parameters.
- static msg_t chDelegateCallDirect4 (thread_t ∗tp, delegate_fn4_t func, msg_t p1, msg_t p2, msg_t p3, msg_t p4)

  Direct call to a function with four parameters.

6.4.3.5.2 Typedef Documentation

6.4.3.5.2.1 delegate_veneer_t

```
typedef msg_t(* delegate_veneer_t) (va_list *argsp)
```

Type of a delegate veneer function.

6.4.3.5.2.2 delegate_fn0_t

```
typedef msg_t(* delegate_fn0_t) (void)
```

Type of a delegate function with no parameters.

6.4.3.5.2.3 delegate_fn1_t

```
typedef msg_t(* delegate_fn1_t) (msg_t p1)
```

Type of a delegate function with one parameter.

6.4.3.5.2.4 delegate_fn2_t

```
typedef msg_t(* delegate_fn2_t) (msg_t p1, msg_t p2)
```

Type of a delegate function with two parameters.

6.4.3.5.2.5 delegate_fn3_t

```
typedef msg_t(* delegate_fn3_t) (msg_t p1, msg_t p2, msg_t p3)
```

Type of a delegate function with three parameters.

6.4.3.5.2.6 delegate_fn4_t

```
typedef msg_t(* delegate_fn4_t) (msg_t p1, msg_t p2, msg_t p3, msg_t p4)
```

Type of a delegate function with four parameters.

6.4.3.5.3 Function Documentation

6.4.3.5.3.1 __ch_delegate_fn0()

```
msg_t __ch_delegate_fn0 (
              va_list * argsp)
```

Veneer for functions with no parameters.

Parameters

in	*argsp*	the list of arguments

Returns

The function return value.

6.4.3.5.3.2 __ch_delegate_fn1()

```
msg_t __ch_delegate_fn1 (
              va_list * argsp)
```

Veneer for functions with one parameter.

Parameters

in	*argsp*	the list of arguments

Returns

The function return value.

6.4.3.5.3.3 __ch_delegate_fn2()

```
msg_t __ch_delegate_fn2 (
              va_list * argsp)
```

Veneer for functions with two parameters.

Parameters

in	*argsp*	the list of arguments

Returns

The function return value.

6.4.3.5.3.4 __ch_delegate_fn3()

```
msg_t __ch_delegate_fn3 (
              va_list * argsp)
```

Veneer for functions with three parameters.

Parameters

in	*argsp*	the list of arguments

Returns

The function return value.

6.4.3.5.3.5 __ch_delegate_fn4()

```
msg_t __ch_delegate_fn4 (
              va_list * argsp)
```

Veneer for functions with four parameters.

Parameters

in	*argsp*	the list of arguments

Returns

The function return value.

6.4.3.5.3.6 chDelegateCallVeneer()

```
msg_t chDelegateCallVeneer (
              thread_t * tp,
              delegate_veneer_t veneer,
               ...)
```

Triggers a function call on a delegate thread.

Note

The thread must be executing `chDelegateDispatchTimeout()` in order to have the functions called.

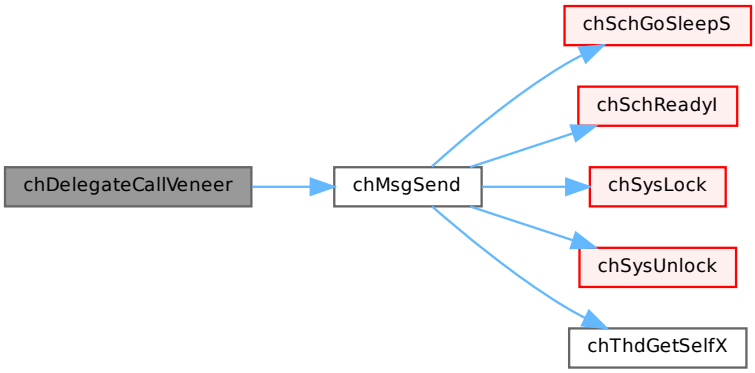
Parameters

in	*tp*	pointer to the delegate thread
in	*veneer*	pointer to the veneer function to be called
in	...	variable number of parameters

Returns

The function return value casted to msg_t. It is garbage for functions returning `void`.

Here is the call graph for this function:



6.4.3.5.3.7 chDelegateDispatch()

```
void chDelegateDispatch (
             void )
```

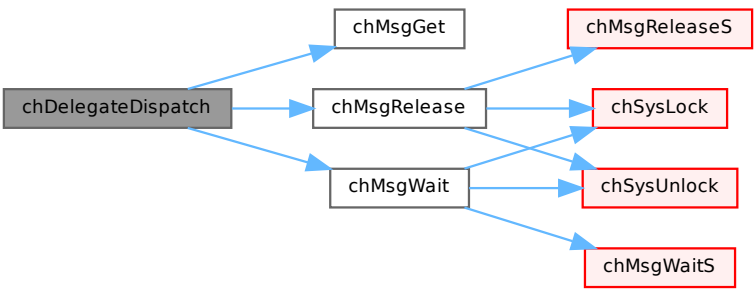
Call messages dispatching.

The function awaits for an incoming call messages and calls the specified functions, then it returns. In case multiple threads are sending messages then the requests are served in priority order.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



6.4.3.5.3.8 chDelegateDispatchTimeout()

```
msg_t chDelegateDispatchTimeout (
            sysinterval_t timeout)
```

Call messages dispatching with timeout.

The function awaits for an incoming call messages and calls the specified functions, then it returns. In case multiple threads are sending messages then the requests are served in priority order.

Parameters

in	*timeout*	the number of ticks before the operation timeouts, the following special values are allowed: • *TIME_INFINITE* no timeout.

Returns

The function outcome.

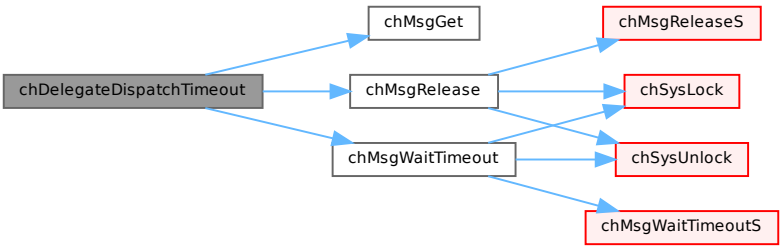
Return values

MSG_OK	if a function has been called.
MSG_TIMEOUT	if a timeout occurred.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



6.4.3.5.3.9 chDelegateCallDirect0()

```
msg_t chDelegateCallDirect0 (
            thread_t * tp,
            delegate_fn0_t func)  [inline], [static]
```

Direct call to a function with no parameters.

Note

The return value is assumed to be not larger than a data pointer type. If you need a portable function then use `chDelegateCallVeneer()` instead.

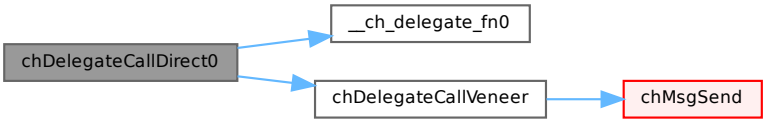
Parameters

in	*tp*	pointer to the delegate thread
in	*func*	pointer to the function to be called

Returns

The function return value as a msg_t.

Here is the call graph for this function:



6.4.3.5.3.10 chDelegateCallDirect1()

```
msg_t chDelegateCallDirect1 (
            thread_t * tp,
            delegate_fn1_t func,
            msg_t p1)  [inline], [static]
```

Direct call to a function with one parameter.

Note

The return value and parameters are assumed to be not larger than a data pointer type. If you need a portable function then use chDelegateCallVeneer() instead.

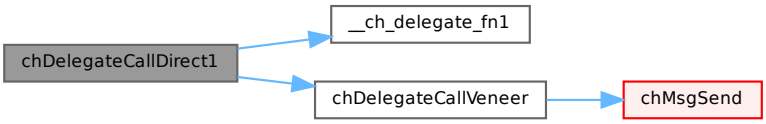
Parameters

in	*tp*	pointer to the delegate thread
in	*func*	pointer to the function to be called
in	*p1*	parameter 1 passed as a msg_t

Returns

The function return value as a msg_t.

Here is the call graph for this function:



6.4.3.5.3.11 chDelegateCallDirect2()

```
msg_t chDelegateCallDirect2 (
            thread_t * tp,
            delegate_fn2_t func,
            msg_t p1,
            msg_t p2)  [inline], [static]
```

Direct call to a function with two parameters.

Note

The return value and parameters are assumed to be not larger than a data pointer type. If you need a portable function then use `chDelegateCallVeneer()` instead.

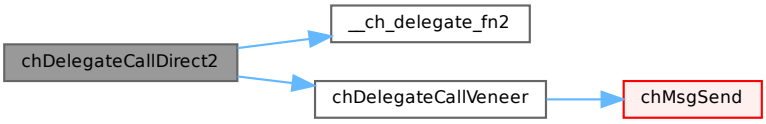
Parameters

in	*tp*	pointer to the delegate thread
in	*func*	pointer to the function to be called
in	*p1*	parameter 1 passed as a `msg_t`
in	*p2*	parameter 2 passed as a `msg_t`

Returns

The function return value as a `msg_t`.

Here is the call graph for this function:



6.4.3.5.3.12 chDelegateCallDirect3()

```
msg_t chDelegateCallDirect3 (
            thread_t * tp,
            delegate_fn3_t func,
            msg_t p1,
            msg_t p2,
            msg_t p3)  [inline], [static]
```

Direct call to a function with three parameters.

Note

The return value and parameters are assumed to be not larger than a data pointer type. If you need a portable function then use `chDelegateCallVeneer()` instead.

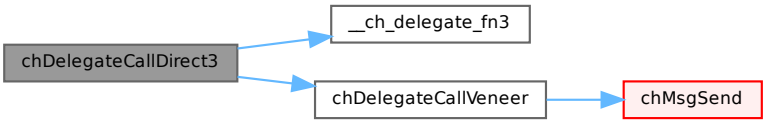
Parameters

in	*tp*	pointer to the delegate thread
in	*func*	pointer to the function to be called
in	*p1*	parameter 1 passed as a msg_t
in	*p2*	parameter 2 passed as a msg_t
in	*p3*	parameter 3 passed as a msg_t

Returns

The function return value as a msg_t.

Here is the call graph for this function:



6.4.3.5.3.13 chDelegateCallDirect4()

```
msg_t chDelegateCallDirect4 (
              thread_t * tp,
              delegate_fn4_t func,
              msg_t p1,
              msg_t p2,
              msg_t p3,
              msg_t p4)  [inline], [static]
```

Direct call to a function with four parameters.

Note

The return value and parameters are assumed to be not larger than a data pointer type. If you need a portable function then use chDelegateCallVeneer() instead.

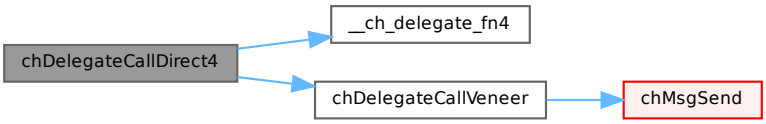
Parameters

in	*tp*	pointer to the delegate thread
in	*func*	pointer to the function to be called
in	*p1*	parameter 1 passed as a msg_t
in	*p2*	parameter 2 passed as a msg_t
in	*p3*	parameter 3 passed as a msg_t
in	*p4*	parameter 4 passed as a msg_t

Returns

The function return value as a msg_t.

Here is the call graph for this function:



6.4.3.6 Jobs Queues

6.4.3.6.1 Detailed Description

Data Structures

- struct ch_jobs_queue
  Type of a jobs queue.
- struct ch_job_descriptor
  Type of a job descriptor.

Macros

- #define MSG_JOB_NULL ((msg_t)-2)
  Dispatcher return code in case of a JOB_NUL has been received.

Typedefs

- typedef struct ch_jobs_queue jobs_queue_t
  Type of a jobs queue.
- typedef void(∗ job_function_t) (void ∗arg)
  Type of a job function.
- typedef struct ch_job_descriptor job_descriptor_t
  Type of a job descriptor.

Functions

- static void chJobObjectInit (jobs_queue_t *jqp, size_t jobsn, job_descriptor_t *jobsbuf, msg_t *msgbuf)
  Initializes a jobs queue object.
- static job_descriptor_t * chJobGet (jobs_queue_t *jqp)
  Allocates a free job object.
- static job_descriptor_t * chJobGetI (jobs_queue_t *jqp)
  Allocates a free job object.
- static job_descriptor_t * chJobGetTimeoutS (jobs_queue_t *jqp, sysinterval_t timeout)
  Allocates a free job object.
- static job_descriptor_t * chJobGetTimeout (jobs_queue_t *jqp, sysinterval_t timeout)
  Allocates a free job object.
- static void chJobPostI (jobs_queue_t *jqp, job_descriptor_t *jp)
  Posts a job object.
- static void chJobPostS (jobs_queue_t *jqp, job_descriptor_t *jp)
  Posts a job object.
- static void chJobPost (jobs_queue_t *jqp, job_descriptor_t *jp)
  Posts a job object.
- static void chJobPostAheadI (jobs_queue_t *jqp, job_descriptor_t *jp)
  Posts an high priority job object.
- static void chJobPostAheadS (jobs_queue_t *jqp, job_descriptor_t *jp)
  Posts an high priority job object.
- static void chJobPostAhead (jobs_queue_t *jqp, job_descriptor_t *jp)
  Posts an high priority job object.
- static msg_t chJobDispatch (jobs_queue_t *jqp)
  Waits for a job then executes it.
- static msg_t chJobDispatchTimeout (jobs_queue_t *jqp, sysinterval_t timeout)
  Waits for a job then executes it.

6.4.3.6.2 Macro Definition Documentation

6.4.3.6.2.1 MSG_JOB_NULL

```
#define MSG_JOB_NULL ((msg_t)-2)
```

Dispatcher return code in case of a JOB_NUL has been received.

6.4.3.6.3 Typedef Documentation

6.4.3.6.3.1 jobs_queue_t

```
typedef struct ch_jobs_queue jobs_queue_t
```

Type of a jobs queue.

6.4.3.6.3.2 job_function_t

```
typedef void(* job_function_t) (void *arg)
```

Type of a job function.

6.4.3.6.3.3 job_descriptor_t

```
typedef struct ch_job_descriptor job_descriptor_t
```

Type of a job descriptor.

6.4.3.6.4 Function Documentation

6.4.3.6.4.1 chJobObjectInit()

```
void chJobObjectInit (
            jobs_queue_t * jqp,
            size_t jobsn,
            job_descriptor_t * jobsbuf,
            msg_t * msgbuf)  [inline], [static]
```

Initializes a jobs queue object.

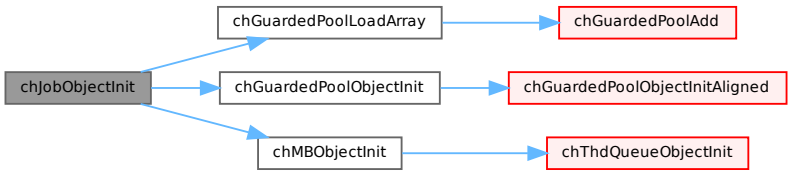
Parameters

out	*jqp*	pointer to a jobs_queue_t structure
in	*jobsn*	number of jobs available
in	*jobsbuf*	pointer to the buffer of jobs, it must be able to hold jobsn job_descriptor_t structures
in	*msgbuf*	pointer to the buffer of messages, it must be able to hold jobsn msg_t messages

Function Class:

Object or module initializer function.

Here is the call graph for this function:



6.4.3.6.4.2 chJobGet()

```
job_descriptor_t * chJobGet (
            jobs_queue_t * jqp)  [inline], [static]
```

Allocates a free job object.

Parameters

in	*jqp*	pointer to a jobs_queue_t structure

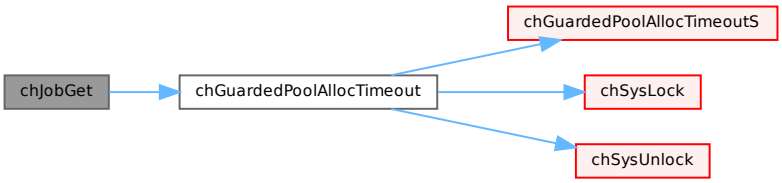
Returns

The pointer to the allocated job object.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



6.4.3.6.4.3 chJobGetI()

```
job_descriptor_t * chJobGetI (
            jobs_queue_t * jqp)  [inline], [static]
```

Allocates a free job object.

Parameters

in	*jqp*	pointer to a jobs_queue_t structure

Returns

The pointer to the allocated job object.

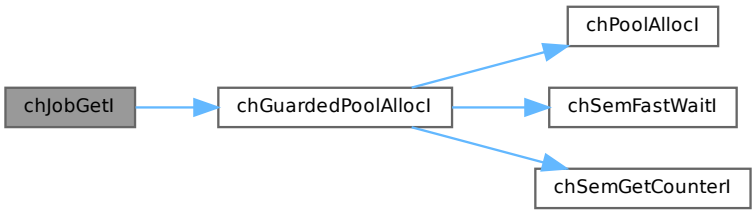
Return values

NULL	if a job object is not immediately available.

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:



6.4.3.6.4.4 chJobGetTimeoutS()

```
job_descriptor_t * chJobGetTimeoutS (
            jobs_queue_t * jqp,
            sysinterval_t timeout)  [inline], [static]
```

Allocates a free job object.

Parameters

in	*jqp*	pointer to a `jobs_queue_t` structure
in	*timeout*	the number of ticks before the operation timeouts, the following special values are allowed: • *TIME_IMMEDIATE* immediate timeout. • *TIME_INFINITE* no timeout.

Returns

The pointer to the allocated job object.

Return values

NULL	if a job object is not available within the specified timeout.

Function Class:

This is an **S-Class** API, this function can be invoked from within a system lock zone by threads only.

Here is the call graph for this function:

6.4.3.6.4.5 chJobGetTimeout()

```
job_descriptor_t * chJobGetTimeout (
            jobs_queue_t * jqp,
            sysinterval_t timeout)  [inline], [static]
```

Allocates a free job object.

Parameters

in	*jqp*	pointer to a jobs_queue_t structure
in	*timeout*	the number of ticks before the operation timeouts, the following special values are allowed: • *TIME_IMMEDIATE* immediate timeout. • *TIME_INFINITE* no timeout.

Returns

The pointer to the allocated job object.

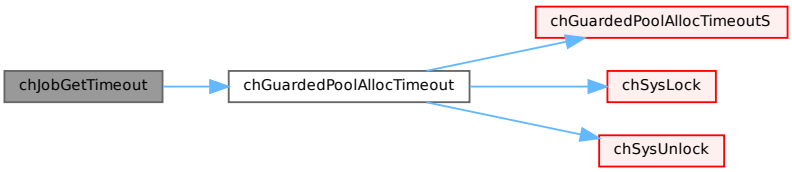
Return values

NULL	if a job object is not available within the specified timeout.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



6.4.3.6.4.6 chJobPostI()

```
void chJobPostI (
            jobs_queue_t * jqp,
            job_descriptor_t * jp)  [inline], [static]
```

Posts a job object.

Note

By design the object can be always immediately posted.

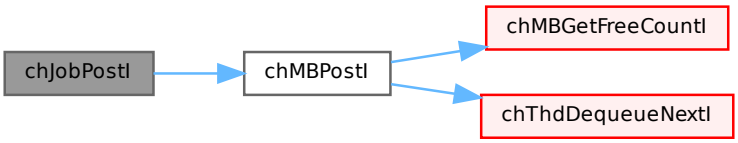
Parameters

in	*jqp*	pointer to a jobs_queue_t structure
in	*jp*	pointer to the job object to be posted

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:



6.4.3.6.4.7 chJobPostS()

```
void chJobPostS (
            jobs_queue_t * jqp,
            job_descriptor_t * jp)  [inline], [static]
```

Posts a job object.

Note

By design the object can be always immediately posted.

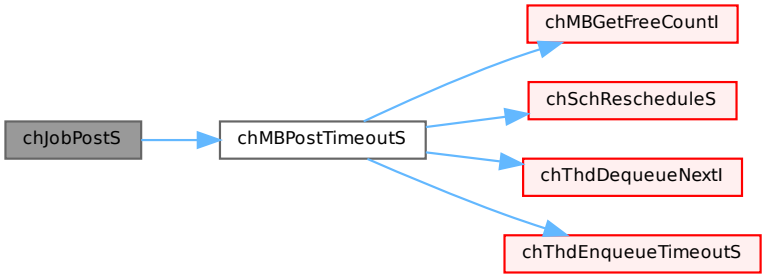
Parameters

in	*jqp*	pointer to a jobs_queue_t structure
in	*jp*	pointer to the job object to be posted

Function Class:

This is an **S-Class** API, this function can be invoked from within a system lock zone by threads only.

Here is the call graph for this function:



6.4.3.6.4.8 chJobPost()

```
void chJobPost (
              jobs_queue_t * jqp,
              job_descriptor_t * jp)  [inline], [static]
```

Posts a job object.

Note

By design the object can be always immediately posted.

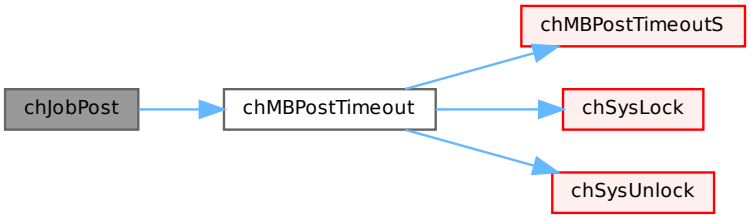
Parameters

in	*jqp*	pointer to a jobs_queue_t structure
in	*jp*	pointer to the job object to be posted

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



6.4.3.6.4.9 chJobPostAheadI()

```
void chJobPostAheadI (
              jobs_queue_t * jqp,
              job_descriptor_t * jp)  [inline], [static]
```

Posts an high priority job object.

Note

By design the object can be always immediately posted.

Parameters

in	*jqp*	pointer to a jobs_queue_t structure
in	*jp*	pointer to the job object to be posted

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:

6.4.3.6.4.10 chJobPostAheadS()

```
void chJobPostAheadS (
             jobs_queue_t * jqp,
             job_descriptor_t * jp)  [inline], [static]
```

Posts an high priority job object.

Note

By design the object can be always immediately posted.

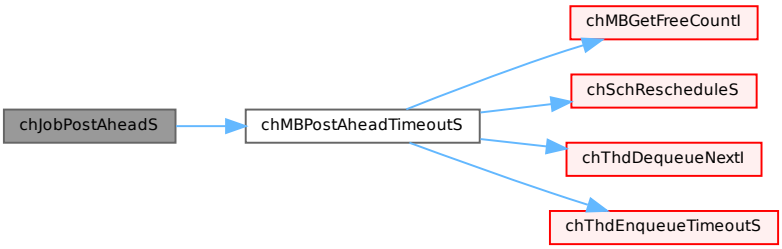
Parameters

in	*jqp*	pointer to a jobs_queue_t structure
in	*jp*	pointer to the job object to be posted

Function Class:

This is an **S-Class** API, this function can be invoked from within a system lock zone by threads only.

Here is the call graph for this function:



6.4.3.6.4.11 chJobPostAhead()

```
void chJobPostAhead (
             jobs_queue_t * jqp,
             job_descriptor_t * jp)  [inline], [static]
```

Posts an high priority job object.

Note

By design the object can be always immediately posted.

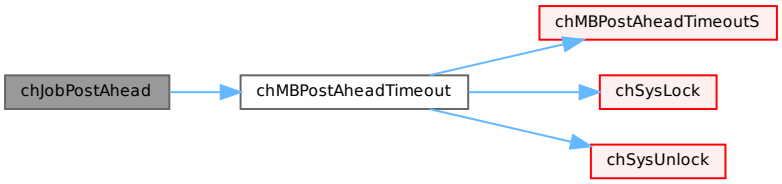
Parameters

in	*jqp*	pointer to a jobs_queue_t structure
in	*jp*	pointer to the job object to be posted

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



6.4.3.6.4.12 chJobDispatch()

```
msg_t chJobDispatch (
              jobs_queue_t * jqp)  [inline], [static]
```

Waits for a job then executes it.

Parameters

in	*jqp*	pointer to a jobs_queue_t structure

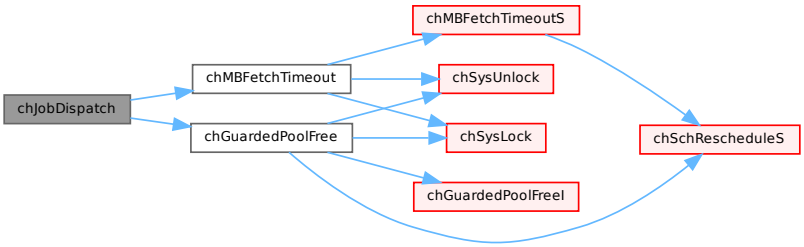
Returns

The function outcome.

Return values

MSG_OK	if a job has been executed.
MSG_RESET	if the internal mailbox has been reset.
MSG_JOB_NULL	if a JOB_NULL has been received.

Here is the call graph for this function:



6.4.3.6.4.13 chJobDispatchTimeout()

```
msg_t chJobDispatchTimeout (
              jobs_queue_t * jqp,
              sysinterval_t timeout)  [inline], [static]
```

Waits for a job then executes it.

Parameters

in	*jqp*	pointer to a jobs_queue_t structure
in	*timeout*	the number of ticks before the operation timeouts, the following special values are allowed: • *TIME_IMMEDIATE* immediate timeout. • *TIME_INFINITE* no timeout.

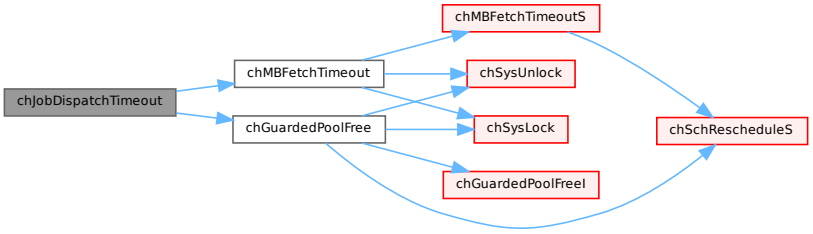
Returns

The function outcome.

Return values

MSG_OK	if a job has been executed.
MSG_TIMEOUT	if a timeout occurred.
MSG_RESET	if the internal mailbox has been reset.
MSG_JOB_NULL	if a JOB_NULL has been received.

Here is the call graph for this function:



6.4.4 Memory Management

6.4.4.1 Detailed Description

Memory Management services.

Topics

- Core Memory Manager
- Memory Heaps
- Memory Pools

6.4.4.2 Core Memory Manager

6.4.4.2.1 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.

Data Structures

- struct memcore_t

  Type of memory core object.

Macros

- #define CH_CFG_MEMCORE_SIZE 0

  Managed RAM size.
- #define chCoreAllocAlignedWithOffsetI chCoreAllocFromTopI

  Allocates a memory block.
- #define chCoreAllocAlignedWithOffset chCoreAllocFromTop

  Allocates a memory block.

Typedefs

- typedef void *(* memgetfunc_t) (size_t size, unsigned align)

  Memory get function.
- typedef void *(* memgetfunc2_t) (size_t size, unsigned align, size_t offset)

  Enhanced memory get function.

Functions

- void __core_init (void)

  Low level memory manager initialization.
- void * chCoreAllocFromBaseI (size_t size, unsigned align, size_t offset)

  Allocates a memory block starting from the lowest address upward.
- void * chCoreAllocFromTopI (size_t size, unsigned align, size_t offset)

  Allocates a memory block starting from the top address downward.
- void * chCoreAllocFromBase (size_t size, unsigned align, size_t offset)

  Allocates a memory block starting from the lowest address upward.
- void * chCoreAllocFromTop (size_t size, unsigned align, size_t offset)

  Allocates a memory block starting from the top address downward.
- size_t chCoreGetStatusX (void)

  Core memory status.
- static void * chCoreAllocAlignedI (size_t size, unsigned align)

  Allocates a memory block.
- static void * chCoreAllocAligned (size_t size, unsigned align)

  Allocates a memory block.
- static void * chCoreAllocI (size_t size)

  Allocates a memory block.
- static void * chCoreAlloc (size_t size)

  Allocates a memory block.

Variables

- memcore_t ch_memcore

  Memory core descriptor.

6.4.4.2.2 Macro Definition Documentation

6.4.4.2.2.1 CH_CFG_MEMCORE_SIZE

```
#define CH_CFG_MEMCORE_SIZE 0
```

Managed RAM size.

Size of the RAM area to be managed by the OS. If set to zero then the whole available RAM is used. The core memory is made available to the heap allocator and/or can be used directly through the simplified core memory allocator.

Note

In order to let the OS manage the whole RAM the linker script must provide the `__heap_base__` and `__heap_end__` symbols.

Requires `CH_CFG_USE_MEMCORE`.

6.4.4.2.2.2 chCoreAllocAlignedWithOffsetI

```
#define chCoreAllocAlignedWithOffsetI chCoreAllocFromTopI
```

Allocates a memory block.

Note

This is a generic form with unspecified allocation position.

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

6.4.4.2.2.3 chCoreAllocAlignedWithOffset

```
#define chCoreAllocAlignedWithOffset chCoreAllocFromTop
```

Allocates a memory block.

Note

This is a generic form with unspecified allocation position.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

6.4.4.2.3 Typedef Documentation

6.4.4.2.3.1 memgetfunc_t

```
typedef void *(* memgetfunc_t) (size_t size, unsigned align)
```

Memory get function.

6.4.4.2.3.2 memgetfunc2_t

```
typedef void *(* memgetfunc2_t) (size_t size, unsigned align, size_t offset)
```

Enhanced memory get function.

6.4.4.2.4 Function Documentation

6.4.4.2.4.1 __core_init()

```
void __core_init (
             void )
```

Low level memory manager initialization.

Function Class:

Not an API, this function is for internal use only.

6.4.4.2.4.2 chCoreAllocFromBaseI()

```
void * chCoreAllocFromBaseI (
             size_t size,
             unsigned align,
             size_t offset)
```

Allocates a memory block starting from the lowest address upward.

This function allocates a block of `offset + size` bytes. The returned pointer has `offset` bytes before its address and `size` bytes after.

Parameters

in	*size*	the size of the block to be allocated.
in	*align*	desired memory alignment
in	*offset*	aligned pointer offset

Returns

A pointer to the allocated memory block.

Return values

NULL	allocation failed, core memory exhausted.

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

6.4.4.2.4.3 chCoreAllocFromTopI()

```
void * chCoreAllocFromTopI (
            size_t size,
            unsigned align,
            size_t offset)
```

Allocates a memory block starting from the top address downward.

This function allocates a block of `offset + size` bytes. The returned pointer has `offset` bytes before its address and `size` bytes after.

Parameters

`in`	*size*	the size of the block to be allocated.
`in`	*align*	desired memory alignment
`in`	*offset*	aligned pointer offset

Returns

A pointer to the allocated memory block.

Return values

NULL	allocation failed, core memory exhausted.

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

6.4.4.2.4.4 chCoreAllocFromBase()

```
void * chCoreAllocFromBase (
            size_t size,
            unsigned align,
            size_t offset)
```

Allocates a memory block starting from the lowest address upward.

This function allocates a block of `offset + size` bytes. The returned pointer has `offset` bytes before its address and `size` bytes after.

Parameters

in	*size*	the size of the block to be allocated.
in	*align*	desired memory alignment
in	*offset*	aligned pointer offset

Returns

A pointer to the allocated memory block.

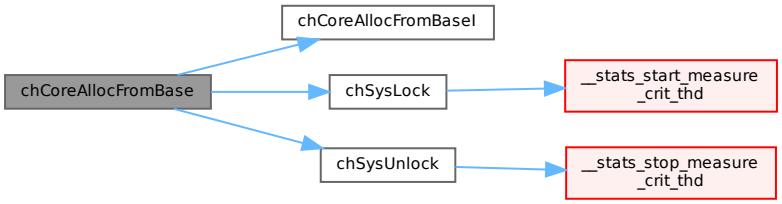
Return values

NULL	allocation failed, core memory exhausted.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



6.4.4.2.4.5 chCoreAllocFromTop()

```
void * chCoreAllocFromTop (
            size_t size,
            unsigned align,
            size_t offset)
```

Allocates a memory block starting from the top address downward.

This function allocates a block of `offset + size` bytes. The returned pointer has `offset` bytes before its address and `size` bytes after.

Parameters

in	*size*	the size of the block to be allocated.
in	*align*	desired memory alignment
in	*offset*	aligned pointer offset

Returns

A pointer to the allocated memory block.

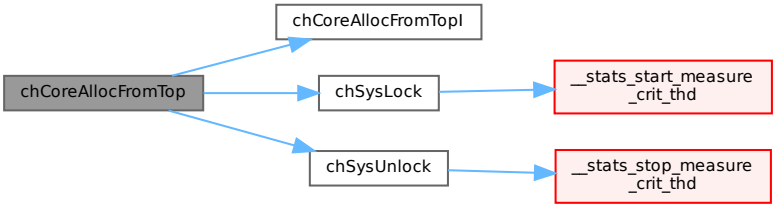
Return values

NULL	allocation failed, core memory exhausted.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



6.4.4.2.4.6 chCoreGetStatusX()

```
size_t chCoreGetStatusX (
            void )
```

Core memory status.

Returns

The size, in bytes, of the free core memory.

Function Class:

This is an **X-Class** API, this function can be invoked from any context.

6.4.4.2.4.7 chCoreAllocAlignedI()

```
void * chCoreAllocAlignedI (
            size_t size,
            unsigned align)  [inline], [static]
```

Allocates a memory block.

The allocated block is guaranteed to be properly aligned to the specified alignment.

Note

This is a generic form with unspecified allocation position.

Parameters

in	*size*	the size of the block to be allocated.
in	*align*	desired memory alignment

Returns

A pointer to the allocated memory block.

Return values

NULL	allocation failed, core memory exhausted.

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

6.4.4.2.4.8 chCoreAllocAligned()

```
void * chCoreAllocAligned (
            size_t size,
            unsigned align)  [inline], [static]
```

Allocates a memory block.

The allocated block is guaranteed to be properly aligned to the specified alignment.

Note

This is a generic form with unspecified allocation position.

Parameters

in	*size*	the size of the block to be allocated
in	*align*	desired memory alignment

Returns

A pointer to the allocated memory block.

Return values

NULL	allocation failed, core memory exhausted.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

6.4.4.2.4.9 chCoreAllocI()

```
void * chCoreAllocI (
            size_t size)  [inline], [static]
```

Allocates a memory block.

The allocated block is guaranteed to be properly aligned for a pointer data type.

Note

This is a generic form with unspecified allocation position.

Parameters

in	*size*	the size of the block to be allocated.

Returns

A pointer to the allocated memory block.

Return values

NULL	allocation failed, core memory exhausted.

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

6.4.4.2.4.10 chCoreAlloc()

```
void * chCoreAlloc (
            size_t size)  [inline], [static]
```

Allocates a memory block.

The allocated block is guaranteed to be properly aligned for a pointer data type.

Note

This is a generic form with unspecified allocation position.

Parameters

in	*size*	the size of the block to be allocated.

Returns

A pointer to the allocated memory block.

Return values

NULL	allocation failed, core memory exhausted.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

6.4.4.2.5 Variable Documentation

6.4.4.2.5.1 ch_memcore

```
memcore_t ch_memcore
```

Memory core descriptor.

6.4.4.3 Memory Heaps

6.4.4.3.1 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.

Data Structures

- union heap_header

  Memory heap block header.
- struct memory_heap

  Structure describing a memory heap.

Macros

- #define H_LOCK(h)
- #define H_UNLOCK(h)
- #define H_BLOCK(hp)
- #define H_LIMIT(hp)
- #define H_NEXT(hp)
- #define H_PAGES(hp)
- #define H_HEAP(hp)
- #define H_SIZE(hp)
- #define NPAGES(p1, p2)
- #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.

Variables

- static memory_heap_t default_heap

  Default heap descriptor.

6.4.4.3.2 Macro Definition Documentation

6.4.4.3.2.1 H_LOCK

```
#define H_LOCK(
               h)
```

Value:

```
chMtxLock(&(h)->mtx)
```

6.4.4.3.2.2 H_UNLOCK

```
#define H_UNLOCK(
               h)
```

Value:

```
chMtxUnlock(&(h)->mtx)
```

6.4.4.3.2.3 H_BLOCK

```
#define H_BLOCK(
               hp)
```

Value:

```
((hp) + 1U)
```

6.4.4.3.2.4 H_LIMIT

```
#define H_LIMIT(
               hp)
```

Value:

```
(H_BLOCK(hp) + H_PAGES(hp))
```

6.4.4.3.2.5 H_NEXT

```
#define H_NEXT(
               hp)
```

Value:

```
((hp)->free.next)
```

6.4.4.3.2.6 H_PAGES

```
#define H_PAGES(
               hp)
```

Value:

```
((hp)->free.pages)
```

6.4.4.3.2.7 H_HEAP

```
#define H_HEAP(
               hp)
```

Value:

```
((hp)->used.heap)
```

6.4.4.3.2.8 H_SIZE

```
#define H_SIZE(
               hp)
```

Value:

```
((hp)->used.size)
```

6.4.4.3.2.9 NPAGES

```
#define NPAGES(
               p1,
               p2)
```

Value:

```
  /*lint -save -e9033 [10.8] The cast is safe.*/                          \
  ((size_t)((p1) - (p2)))                                                 \
  /*lint -restore*/
```

6.4.4.3.2.10 CH_HEAP_ALIGNMENT

```
#define CH_HEAP_ALIGNMENT 8U
```

Minimum alignment used for heap.

Note

Cannot use the sizeof operator in this macro.

6.4.4.3.2.11 CH_HEAP_AREA

```
#define CH_HEAP_AREA(
               name,
               size)
```

Value:

```
  ALIGNED_VAR(CH_HEAP_ALIGNMENT)                                          \
  uint8_t name[MEM_ALIGN_NEXT((size), CH_HEAP_ALIGNMENT)]
```

Allocation of an aligned static heap buffer.

6.4.4.3.3 Typedef Documentation

6.4.4.3.3.1 memory_heap_t

```
typedef struct memory_heap memory_heap_t
```

Type of a memory heap.

6.4.4.3.3.2 heap_header_t

```
typedef union heap_header heap_header_t
```

Type of a memory heap header.

6.4.4.3.4 Function Documentation

6.4.4.3.4.1 __heap_init()

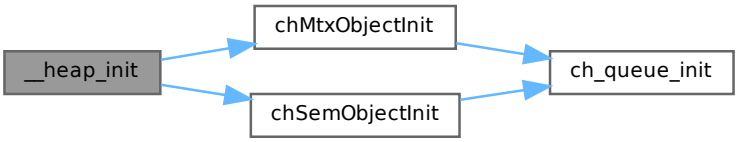
```
void __heap_init (
              void )
```

Initializes the default heap.

Function Class:

Not an API, this function is for internal use only.

Here is the call graph for this function:



6.4.4.3.4.2 chHeapObjectInit()

```
void chHeapObjectInit (
              memory_heap_t * heapp,
              void * buf,
              size_t size)
```

Initializes a memory heap from a static memory area.

Note

The heap buffer base and size are adjusted if the passed buffer is not aligned to `CH_HEAP_ALIGNMENT`. This mean that the effective heap size can be less than `size`.

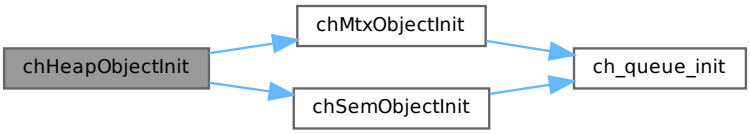
Parameters

out	*heapp*	pointer to the memory heap descriptor to be initialized
in	*buf*	heap buffer base
in	*size*	heap size

Function Class:

Object or module nitializer function.

Here is the call graph for this function:



6.4.4.3.4.3 chHeapAllocAligned()

```
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

in	*heapp*	pointer to a heap descriptor or NULL in order to access the default heap.
in	*size*	the size of the block to be allocated. Note that the allocated block may be a bit bigger than the requested size for alignment and fragmentation reasons.
in	*align*	desired memory alignment

Returns

A pointer to the aligned allocated block.

Return values

NULL	if the block cannot be allocated.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

6.4.4.3.4.4 chHeapFree()

```
void chHeapFree (
            void * p)
```

Frees a previously allocated memory block.

Parameters

in	*p*	pointer to the memory block to be freed

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

6.4.4.3.4.5 chHeapStatus()

```
size_t chHeapStatus (
            memory_heap_t * heapp,
            size_t * totalp,
            size_t * largestp)
```

Reports the heap status.

Note

This function is meant to be used in the test suite, it should not be really useful for the application code.

Parameters

in	*heapp*	pointer to a heap descriptor or NULL in order to access the default heap.
in	*totalp*	pointer to a variable that will receive the total fragmented free space or NULL
in	*largestp*	pointer to a variable that will receive the largest free free block found space or NULL

Returns

The number of fragments in the heap.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

6.4.4.3.4.6 chHeapAlloc()

```
void * chHeapAlloc (
            memory_heap_t * heapp,
            size_t size)  [inline], [static]
```

Allocates a block of memory from the heap by using the first-fit algorithm.

The allocated block is guaranteed to be properly aligned for a pointer data type.

Parameters

in	*heapp*	pointer to a heap descriptor or NULL in order to access the default heap.
in	*size*	the size of the block to be allocated. Note that the allocated block may be a bit bigger than the requested size for alignment and fragmentation reasons.

Returns

A pointer to the allocated block.

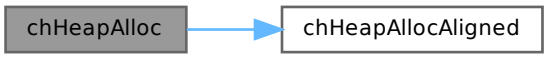
Return values

NULL	if the block cannot be allocated.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



6.4.4.3.4.7 chHeapGetSize()

```
size_t chHeapGetSize (
            const void * p)  [inline], [static]
```

Returns the size of an allocated block.

Note

The returned value is the requested size, the real size is the same value aligned to the next CH_HEAP_↩ ALIGNMENT multiple.

Parameters

in	*p*	pointer to the memory block

Returns

Size of the block.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

6.4.4.3.5 Variable Documentation

6.4.4.3.5.1 default_heap

```
memory_heap_t default_heap  [static]
```

Default heap descriptor.

6.4.4.4 Memory Pools

6.4.4.4.1 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.

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)
  Data part of a static memory pool initializer.
- #define MEMORYPOOL_DECL(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)
  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 * chPoolAllocI (memory_pool_t *mp)

  Allocates an object from a memory pool.
- void * chPoolAlloc (memory_pool_t *mp)

  Allocates an object from a memory pool.
- void chPoolFreeI (memory_pool_t *mp, void *objp)

  Releases an object into a memory pool.
- void chPoolFree (memory_pool_t *mp, void *objp)

  Releases an object into a memory pool.
- void chGuardedPoolObjectInitAligned (guarded_memory_pool_t *gmp, size_t size, unsigned align)

  Initializes an empty guarded memory pool.
- void chGuardedPoolLoadArray (guarded_memory_pool_t *gmp, void *p, size_t n)

  Loads a guarded memory pool with an array of static objects.
- void * chGuardedPoolAllocTimeoutS (guarded_memory_pool_t *gmp, sysinterval_t timeout)

  Allocates an object from a guarded memory pool.
- void * chGuardedPoolAllocTimeout (guarded_memory_pool_t *gmp, sysinterval_t timeout)

  Allocates an object from a guarded memory pool.
- void chGuardedPoolFree (guarded_memory_pool_t *gmp, void *objp)

  Releases an object into a guarded memory pool.
- static void chPoolObjectInit (memory_pool_t *mp, size_t size, memgetfunc_t provider)

  Initializes an empty memory pool.
- static void chPoolAdd (memory_pool_t *mp, void *objp)

  Adds an object to a memory pool.
- static void chPoolAddI (memory_pool_t *mp, void *objp)

  Adds an object to a memory pool.
- static void chGuardedPoolObjectInit (guarded_memory_pool_t *gmp, size_t size)

  Initializes an empty guarded memory pool.
- static cnt_t chGuardedPoolGetCounterI (guarded_memory_pool_t *gmp)

  Gets the count of objects in a guarded memory pool.
- static void * chGuardedPoolAllocI (guarded_memory_pool_t *gmp)

  Allocates an object from a guarded memory pool.
- static void chGuardedPoolFreeI (guarded_memory_pool_t *gmp, void *objp)

  Releases an object into a guarded memory pool.
- static void chGuardedPoolFreeS (guarded_memory_pool_t *gmp, void *objp)

  Releases an object into a guarded memory pool.
- static void chGuardedPoolAdd (guarded_memory_pool_t *gmp, void *objp)

  Adds an object to a guarded memory pool.
- static void chGuardedPoolAddI (guarded_memory_pool_t *gmp, void *objp)

  Adds an object to a guarded memory pool.
- static void chGuardedPoolAddS (guarded_memory_pool_t *gmp, void *objp)

  Adds an object to a guarded memory pool.

6.4.4.4.2 Macro Definition Documentation

6.4.4.4.2.1 __MEMORYPOOL_DATA

```
#define __MEMORYPOOL_DATA(
              name,
              size,
              align,
              provider)
```

Value:

```
{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

in	*name*	the name of the memory pool variable
in	*size*	size of the memory pool contained objects
in	*align*	required memory alignment
in	*provider*	memory provider function for the memory pool

6.4.4.4.2.2 MEMORYPOOL_DECL

```
#define MEMORYPOOL_DECL(
              name,
              size,
              align,
              provider)
```

Value:

```
memory_pool_t name = __MEMORYPOOL_DATA(name, size, align, provider)
```

Static memory pool initializer.

Statically initialized memory pools require no explicit initialization using `chPoolInit()`.

Parameters

in	*name*	the name of the memory pool variable
in	*size*	size of the memory pool contained objects
in	*align*	required memory alignment
in	*provider*	memory provider function for the memory pool or `NULL` if the pool is not allowed to grow automatically

6.4.4.4.2.3 __GUARDEDMEMORYPOOL_DATA

```
#define __GUARDEDMEMORYPOOL_DATA(
              name,
              size,
              align)
```

Value:

```
  {                         \
  __SEMAPHORE_DATA(name.sem, (cnt_t)0),                                 \
  __MEMORYPOOL_DATA(NULL, size, align, NULL)                            \
}
```

Data part of a static guarded memory pool initializer.

This macro should be used when statically initializing a memory pool that is part of a bigger structure.

Parameters

in	*name*	the name of the memory pool variable
in	*size*	size of the memory pool contained objects
in	*align*	required memory alignment

6.4.4.4.2.4 GUARDEDMEMORYPOOL_DECL

```
#define GUARDEDMEMORYPOOL_DECL(
              name,
              size,
              align)
```

Value:

```
  guarded_memory_pool_t name = __GUARDEDMEMORYPOOL_DATA(name, size, align)
```

Static guarded memory pool initializer.

Statically initialized guarded memory pools require no explicit initialization using `chGuardedPoolInit()`.

Parameters

in	*name*	the name of the guarded memory pool variable
in	*size*	size of the memory pool contained objects
in	*align*	required memory alignment

6.4.4.4.3 Function Documentation

6.4.4.4.3.1 chPoolObjectInitAligned()

```
void chPoolObjectInitAligned (
             memory_pool_t * mp,
             size_t size,
             unsigned align,
             memgetfunc_t provider)
```

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	*align*	required memory alignment
in	*provider*	memory provider function for the memory pool or NULL if the pool is not allowed to grow automatically

Function Class:

Object or module nitializer function.

6.4.4.4.3.2 chPoolLoadArray()

```
void chPoolLoadArray (
              memory_pool_t * mp,
              void * p,
              size_t n)
```

Loads a memory pool with an array of static objects.

Precondition

The memory pool must already be initialized.

The array elements must be of the right size for the specified memory pool.

The array elements size must be a multiple of the alignment requirement for the pool.

Postcondition

The memory pool contains the elements of the input array.

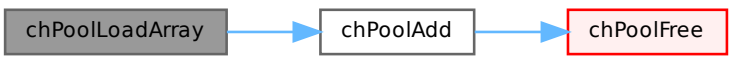
Parameters

in	*mp*	pointer to a memory_pool_t structure
in	*p*	pointer to the array first element
in	*n*	number of elements in the array

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



6.4.4.4.3.3 chPoolAllocI()

```
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.

6.4.4.4.3.4 chPoolAlloc()

```
void * chPoolAlloc (
            memory_pool_t * mp)
```

Allocates an object from a memory pool.

Precondition

The memory pool must already be initialized.

Parameters

in	*mp*	pointer to a memory_pool_t structure

Returns

The pointer to the allocated object.

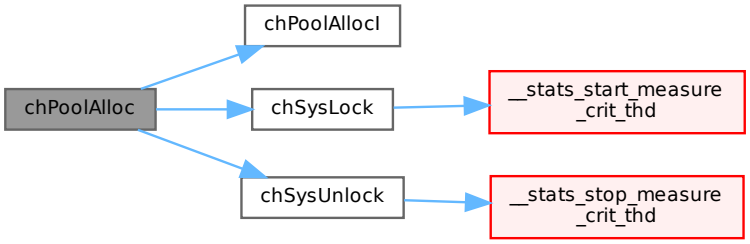
Return values

NULL	if pool is empty.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



6.4.4.4.3.5 chPoolFreeI()

```
void chPoolFreeI (
             memory_pool_t * mp,
             void * objp)
```

Releases an object into a memory pool.

Precondition

The memory pool must already be initialized.

The freed object must be of the right size for the specified memory pool.

The added object must be properly aligned.

Parameters

in	*mp*	pointer to a memory_pool_t structure
in	*objp*	the pointer to the object to be released

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

6.4.4.4.3.6 chPoolFree()

```
void chPoolFree (
              memory_pool_t * mp,
              void * objp)
```

Releases an object into a memory pool.

Precondition

The memory pool must already be initialized.

The freed object must be of the right size for the specified memory pool.

The added object must be properly aligned.

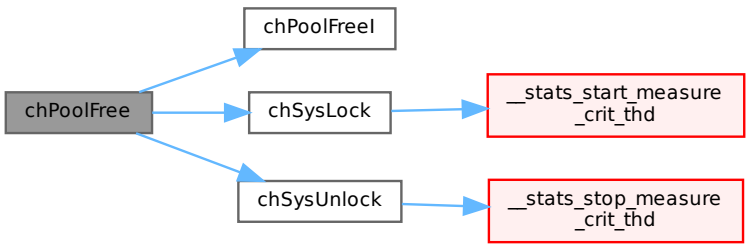
Parameters

in	*mp*	pointer to a 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:



6.4.4.4.3.7 chGuardedPoolObjectInitAligned()

```
void chGuardedPoolObjectInitAligned (
              guarded_memory_pool_t * gmp,
              size_t size,
              unsigned align)
```

Initializes an empty guarded memory pool.

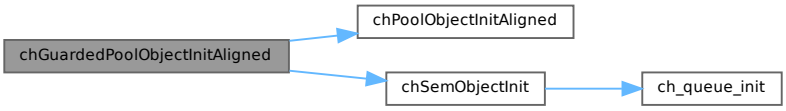
Parameters

out	*gmp*	pointer to a guarded_memory_pool_t structure
in	*size*	the size of the objects contained in this guarded memory pool, the minimum accepted size is the size of a pointer to void.
in	*align*	required memory alignment

Function Class:

Object or module nitializer function.

Here is the call graph for this function:



6.4.4.4.3.8 chGuardedPoolLoadArray()

```
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.

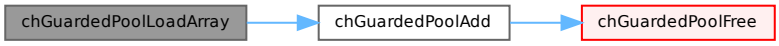
Parameters

in	*gmp*	pointer to a guarded_memory_pool_t structure
in	*p*	pointer to the array first element
in	*n*	number of elements in the array

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



6.4.4.4.3.9 chGuardedPoolAllocTimeoutS()

```
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

in	*gmp*	pointer to a guarded_memory_pool_t structure
in	*timeout*	the number of ticks before the operation timeouts, the following special values are allowed: • *TIME_IMMEDIATE* immediate timeout. • *TIME_INFINITE* no timeout.

Returns

The pointer to the allocated object.

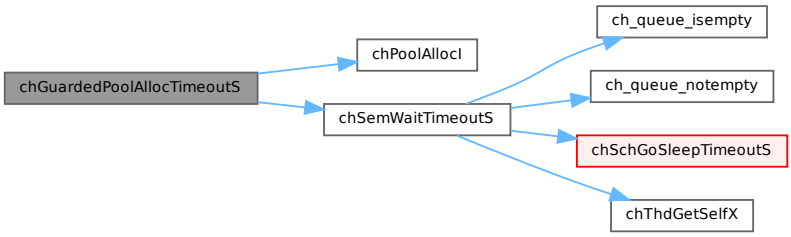
Return values

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:



6.4.4.4.3.10 chGuardedPoolAllocTimeout()

```
void * chGuardedPoolAllocTimeout (
            guarded_memory_pool_t * gmp,
            sysinterval_t timeout)
```

Allocates an object from a guarded memory pool.

Precondition

The guarded memory pool must already be initialized.

Parameters

in	*gmp*	pointer to a guarded_memory_pool_t structure
in	*timeout*	the number of ticks before the operation timeouts, the following special values are allowed: • *TIME_IMMEDIATE* immediate timeout. • *TIME_INFINITE* no timeout.

Returns

The pointer to the allocated object.

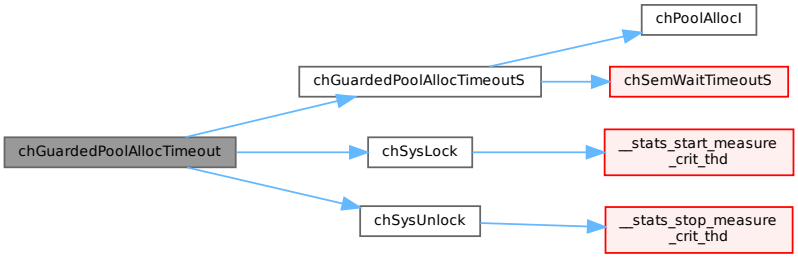
Return values

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:



6.4.4.4.3.11 chGuardedPoolFree()

```
void chGuardedPoolFree (
              guarded_memory_pool_t * gmp,
              void * objp)
```

Releases an object into a guarded memory pool.

Precondition

The guarded memory pool must already be initialized.

The freed object must be of the right size for the specified guarded memory pool.

The added object must be properly aligned.

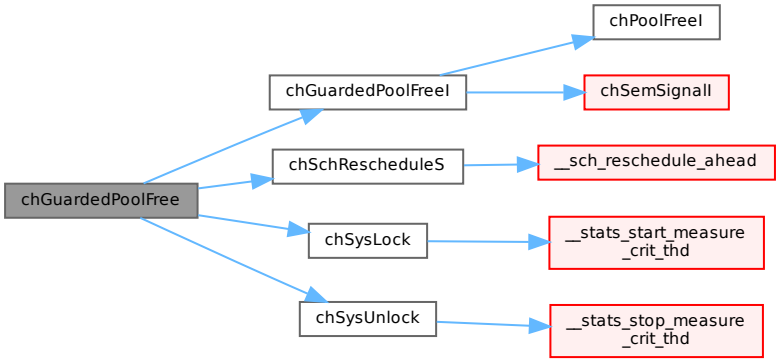
Parameters

in	*gmp*	pointer to a guarded_memory_pool_t structure
in	*objp*	the pointer to the object to be released

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



6.4.4.4.3.12 chPoolObjectInit()

```
void chPoolObjectInit (
              memory_pool_t * mp,
              size_t size,
              memgetfunc_t provider)  [inline], [static]
```

Initializes an empty memory pool.

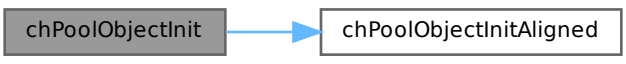
Parameters

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:

Object or module nitializer function.

Here is the call graph for this function:



6.4.4.4.3.13 chPoolAdd()

```
void chPoolAdd (
              memory_pool_t * mp,
              void * objp)  [inline], [static]
```

Adds an object to a memory pool.

Precondition

The memory pool must be already been initialized.

The added object must be of the right size for the specified memory pool.

The added object must be properly aligned.

Note

This function is just an alias for `chPoolFree()` and has been added for clarity.

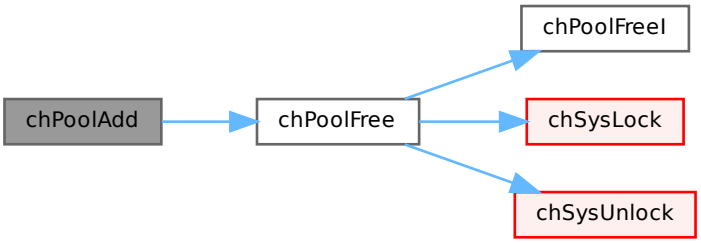
Parameters

in	*mp*	pointer to a `memory_pool_t` structure
in	*objp*	the pointer to the object to be added

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



6.4.4.4.3.14 chPoolAddI()

```
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.

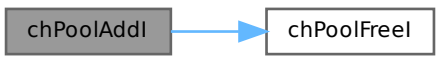
Parameters

in	*mp*	pointer to a `memory_pool_t` structure
in	*objp*	the pointer to the object to be added

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:



6.4.4.4.3.15 chGuardedPoolObjectInit()

```
void chGuardedPoolObjectInit (
            guarded_memory_pool_t * gmp,
            size_t size)  [inline], [static]
```

Initializes an empty guarded memory pool.

Parameters

out	*gmp*	pointer to a guarded_memory_pool_t structure
in	*size*	the size of the objects contained in this guarded memory pool, the minimum accepted size is the size of a pointer to void.

Function Class:

Object or module nitializer function.

Here is the call graph for this function:

6.4.4.4.3.16 chGuardedPoolGetCounterI()

```
cnt_t chGuardedPoolGetCounterI (
            guarded_memory_pool_t * gmp)  [inline], [static]
```

Gets the count of objects in a guarded memory pool.

Precondition

The guarded memory pool must be already been initialized.

Parameters

in	*gmp*	pointer to a guarded_memory_pool_t structure

Returns

The counter of the guard semaphore.

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:

6.4.4.4.3.17 chGuardedPoolAllocI()

```
void * chGuardedPoolAllocI (
            guarded_memory_pool_t * gmp)  [inline], [static]
```

Allocates an object from a guarded memory pool.

Precondition

The guarded memory pool must be already been initialized.

Parameters

in	*gmp*	pointer to a guarded_memory_pool_t structure

Returns

The pointer to the allocated object.

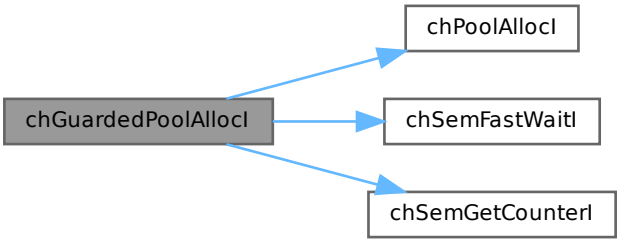
Return values

NULL	if the pool is empty.

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:



6.4.4.4.3.18 chGuardedPoolFreeI()

```
void chGuardedPoolFreeI (
            guarded_memory_pool_t * gmp,
            void * objp)  [inline], [static]
```

Releases an object into a guarded memory pool.

Precondition

The guarded memory pool must already be initialized.

The freed object must be of the right size for the specified guarded memory pool.

The added object must be properly aligned.

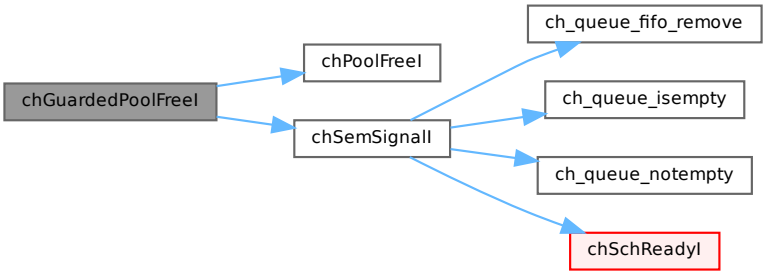
Parameters

in	*gmp*	pointer to a guarded_memory_pool_t structure
in	*objp*	the pointer to the object to be released

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:



6.4.4.4.3.19 chGuardedPoolFreeS()

```
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.

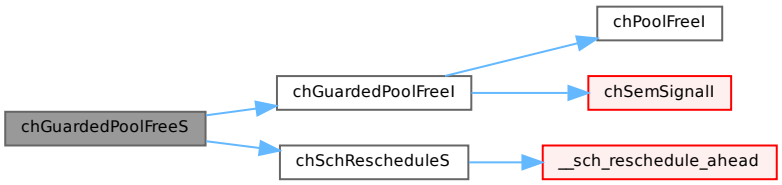
Parameters

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 **S-Class** API, this function can be invoked from within a system lock zone by threads only.

Here is the call graph for this function:



6.4.4.4.3.20 chGuardedPoolAdd()

```
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.

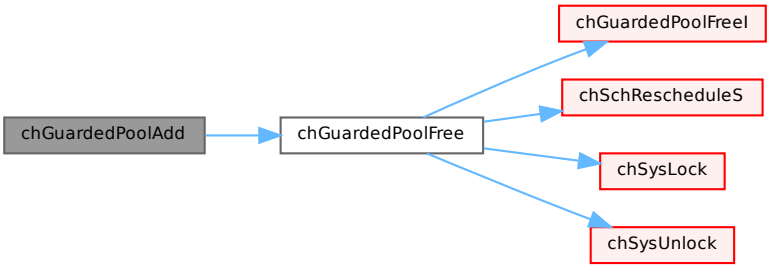
Parameters

in	*gmp*	pointer to a `guarded_memory_pool_t` structure
in	*objp*	the pointer to the object to be added

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



6.4.4.4.3.21 chGuardedPoolAddI()

```
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

in	*gmp*	pointer to a `guarded_memory_pool_t` structure
in	*objp*	the pointer to the object to be added

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:

6.4.4.4.3.22 chGuardedPoolAddS()

```
void chGuardedPoolAddS (
            guarded_memory_pool_t * gmp,
            void * objp)  [inline], [static]
```

Adds an object to a guarded memory pool.

Precondition

The guarded memory pool must be already been initialized.

The added object must be of the right size for the specified guarded memory pool.

The added object must be properly aligned.

Note

This function is just an alias for `chGuardedPoolFreeI()` and has been added for clarity.

Parameters

in	*gmp*	pointer to a `guarded_memory_pool_t` structure
in	*objp*	the pointer to the object to be added

Function Class:

This is an **S-Class** API, this function can be invoked from within a system lock zone by threads only.

Here is the call graph for this function:

6.4.5 Complex Services

6.4.5.1 Detailed Description

Topics

- Objects FIFOs
- Objects Caches
- Dynamic Objects Factory

6.4.5.2 Objects FIFOs

6.4.5.2.1 Detailed Description

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.

6.4.5.2.2 Typedef Documentation

6.4.5.2.2.1 objects_fifo_t

```
typedef struct ch_objects_fifo objects_fifo_t
```

Type of an objects FIFO.

6.4.5.2.3 Function Documentation

6.4.5.2.3.1 chFifoObjectInitAligned()

```
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.

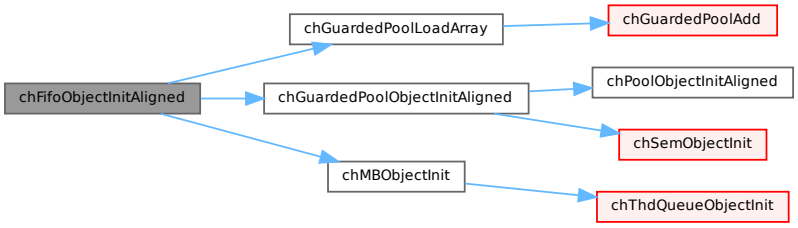
Parameters

out	*ofp*	pointer to a objects_fifo_t structure
in	*objsize*	object size
in	*objn*	number of objects available
in	*objalign*	required objects alignment
in	*objbuf*	pointer to the buffer of objects, it must be able to hold objn objects of objsize size with objalign alignment
in	*msgbuf*	pointer to the buffer of messages, it must be able to hold objn messages

Function Class:

Object or module nitializer function.

Here is the call graph for this function:



6.4.5.2.3.2 chFifoObjectInit()

```
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.

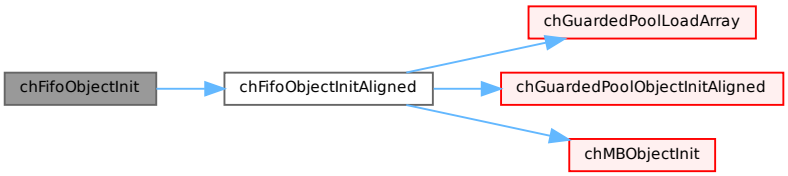
Parameters

out	*ofp*	pointer to a objects_fifo_t structure
in	*objsize*	object size
in	*objn*	number of objects available
in	*objbuf*	pointer to the buffer of objects, it must be able to hold objn objects of objsize size
in	*msgbuf*	pointer to the buffer of messages, it must be able to hold objn messages

Function Class:

Object or module nitializer function.

Here is the call graph for this function:



6.4.5.2.3.3 chFifoTakeObjectI()

```
void * chFifoTakeObjectI (
            objects_fifo_t * ofp)  [inline], [static]
```

Allocates a free object.

Parameters

in	*ofp*	pointer to a objects_fifo_t structure

Returns

The pointer to the allocated object.

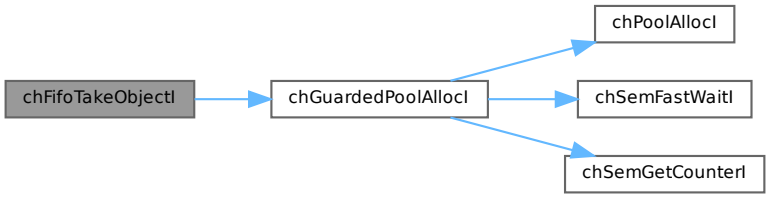
Return values

NULL	if an object is not immediately available.

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:



6.4.5.2.3.4 chFifoTakeObjectTimeoutS()

```
void * chFifoTakeObjectTimeoutS (
            objects_fifo_t * ofp,
            sysinterval_t timeout)  [inline], [static]
```

Allocates a free object.

Parameters

in	*ofp*	pointer to a objects_fifo_t structure
in	*timeout*	the number of ticks before the operation timeouts, the following special values are allowed: • *TIME_IMMEDIATE* immediate timeout. • *TIME_INFINITE* no timeout.

Returns

The pointer to the allocated 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:

6.4.5.2.3.5 chFifoTakeObjectTimeout()

```
void * chFifoTakeObjectTimeout (
            objects_fifo_t * ofp,
            sysinterval_t timeout)  [inline], [static]
```

Allocates a free object.

Parameters

in	*ofp*	pointer to a objects_fifo_t structure
in	*timeout*	the number of ticks before the operation timeouts, the following special values are allowed: • *TIME_IMMEDIATE* immediate timeout. • *TIME_INFINITE* no timeout.

Returns

The pointer to the allocated object.

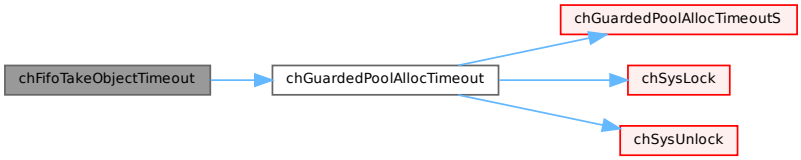
Return values

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:



6.4.5.2.3.6 chFifoReturnObjectI()

```
void chFifoReturnObjectI (
             objects_fifo_t * ofp,
             void * objp)  [inline], [static]
```

Releases a fetched object.

Parameters

in	*ofp*	pointer to a objects_fifo_t structure
in	*objp*	pointer to the object to be released

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:

6.4.5.2.3.7 chFifoReturnObjectS()

```
void chFifoReturnObjectS (
            objects_fifo_t * ofp,
            void * objp)  [inline], [static]
```

Releases a fetched object.

Parameters

in	*ofp*	pointer to a objects_fifo_t structure
in	*objp*	pointer to the object to be released

Function Class:

This is an **S-Class** API, this function can be invoked from within a system lock zone by threads only.

Here is the call graph for this function:

6.4.5.2.3.8 chFifoReturnObject()

```
void chFifoReturnObject (
            objects_fifo_t * ofp,
            void * objp)  [inline], [static]
```

Releases a fetched object.

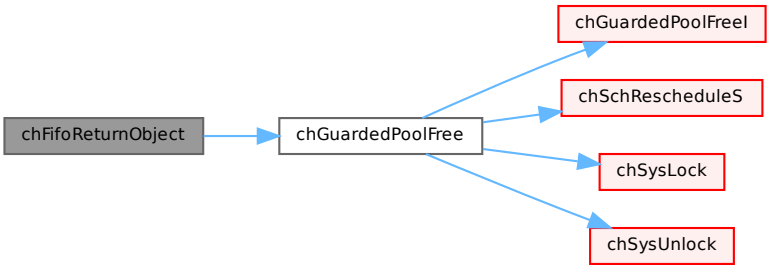
Parameters

in	*ofp*	pointer to a objects_fifo_t structure
in	*objp*	pointer to the object to be released

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



6.4.5.2.3.9 chFifoSendObjectI()

```
void chFifoSendObjectI (
             objects_fifo_t * ofp,
             void * objp)  [inline], [static]
```

Posts an object.

Note

By design the object can be always immediately posted.

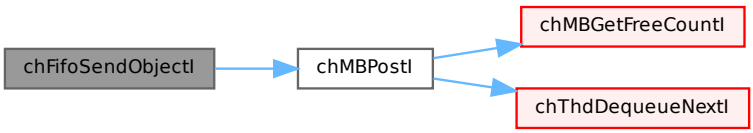
Parameters

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:



6.4.5.2.3.10 chFifoSendObjectS()

```
void chFifoSendObjectS (
              objects_fifo_t * ofp,
              void * objp)  [inline], [static]
```

Posts an object.

Note

By design the object can be always immediately posted.

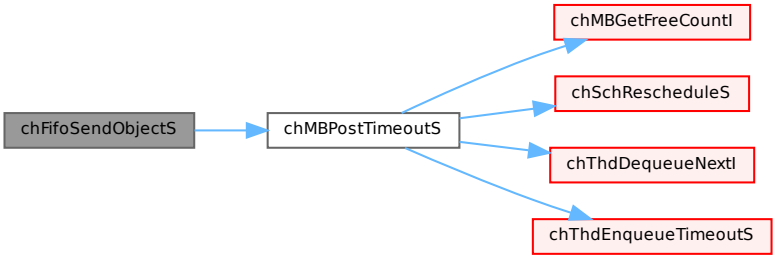
Parameters

in	*ofp*	pointer to a objects_fifo_t structure
in	*objp*	pointer to the object to be posted

Function Class:

This is an **S-Class** API, this function can be invoked from within a system lock zone by threads only.

Here is the call graph for this function:



6.4.5.2.3.11 chFifoSendObject()

```
void chFifoSendObject (
             objects_fifo_t * ofp,
             void * objp)  [inline], [static]
```

Posts an object.

Note

By design the object can be always immediately posted.

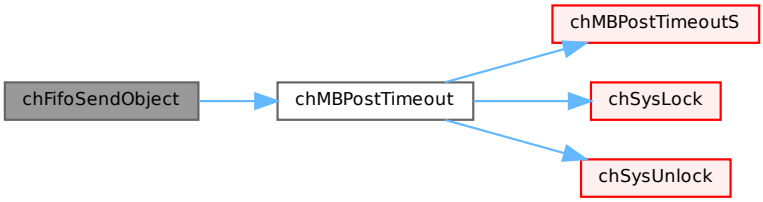
Parameters

in	*ofp*	pointer to a objects_fifo_t structure
in	*objp*	pointer to the object to be released

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



6.4.5.2.3.12 chFifoSendObjectAheadI()

```
void chFifoSendObjectAheadI (
             objects_fifo_t * ofp,
             void * objp)  [inline], [static]
```

Posts an high priority object.

Note

By design the object can be always immediately posted.

Parameters

in	*ofp*	pointer to a objects_fifo_t structure
in	*objp*	pointer to the object to be posted

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:

6.4.5.2.3.13 chFifoSendObjectAheadS()

```
void chFifoSendObjectAheadS (
             objects_fifo_t * ofp,
             void * objp)  [inline], [static]
```

Posts an high priority object.

Note

By design the object can be always immediately posted.

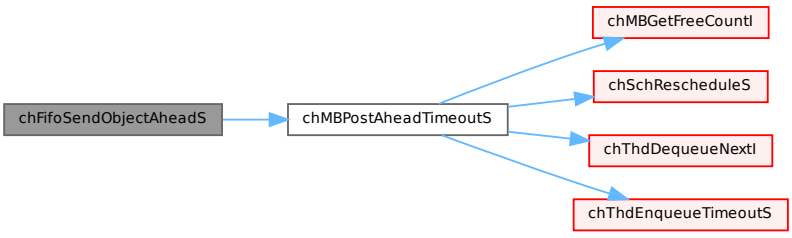
Parameters

in	*ofp*	pointer to a objects_fifo_t structure
in	*objp*	pointer to the object to be posted

Function Class:

This is an **S-Class** API, this function can be invoked from within a system lock zone by threads only.

Here is the call graph for this function:



6.4.5.2.3.14 chFifoSendObjectAhead()

```
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.

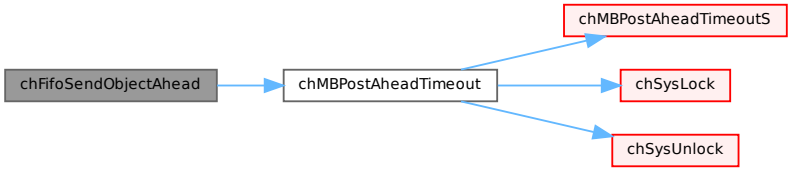
Parameters

in	*ofp*	pointer to a objects_fifo_t structure
in	*objp*	pointer to the object to be released

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



6.4.5.2.3.15 chFifoReceiveObjectI()

```
msg_t chFifoReceiveObjectI (
             objects_fifo_t * ofp,
             void ** objpp)  [inline], [static]
```

Fetches 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:

6.4.5.2.3.16 chFifoReceiveObjectTimeoutS()

```
msg_t chFifoReceiveObjectTimeoutS (
            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.

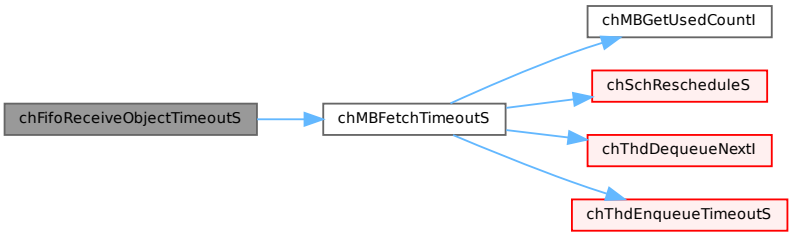
Return values

MSG_OK	if an object has been correctly fetched.
MSG_TIMEOUT	if the operation has timed out.

Function Class:

This is an **S-Class** API, this function can be invoked from within a system lock zone by threads only.

Here is the call graph for this function:



6.4.5.2.3.17 chFifoReceiveObjectTimeout()

```
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.

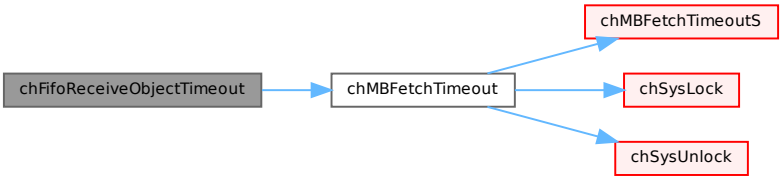
Return values

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:



6.4.5.3 Objects Caches

6.4.5.3.1 Detailed Description

Cached objects flags

- #define OC_FLAG_INLRU 0x00000001U
- #define OC_FLAG_INHASH 0x00000002U
- #define OC_FLAG_SHARED 0x00000004U
- #define OC_FLAG_NOTSYNC 0x00000008U
- #define OC_FLAG_LAZYWRITE 0x00000010U
- #define OC_FLAG_FORGET 0x00000020U

Data Structures

- struct ch_oc_hash_header

  Structure representing an hash table element.
- struct ch_oc_lru_header

  Structure representing an hash table element.
- struct ch_oc_object

  Structure representing a cached object.
- struct ch_objects_cache

  Structure representing a cache object.

Macros

- #define OC_HASH_FUNCTION(ocp, group, key)
- #define HASH_INSERT(ocp, objp, group, key)
- #define HASH_REMOVE(objp)
- #define LRU_INSERT_HEAD(ocp, objp)
- #define LRU_INSERT_TAIL(ocp, objp)
- #define LRU_REMOVE(objp)

Typedefs

- typedef uint32_t oc_flags_t

  Flags of cached objects.
- typedef struct ch_oc_hash_header oc_hash_header_t

  Type of an hash element header.
- typedef struct ch_oc_lru_header oc_lru_header_t

  Type of an LRU element header.
- typedef struct ch_oc_object oc_object_t

  Type of a cached object.
- typedef struct ch_objects_cache objects_cache_t

  Type of a cache object.
- typedef bool(∗ oc_readf_t) (objects_cache_t ∗ocp, oc_object_t ∗objp, bool async)

  Object read function.
- typedef bool(∗ oc_writef_t) (objects_cache_t ∗ocp, oc_object_t ∗objp, bool async)

  Object write function.

Functions

- static oc_object_t ∗ hash_get_s (objects_cache_t ∗ocp, uint32_t group, uint32_t key)

  Returns an object pointer from the cache, if present.
- static oc_object_t ∗ lru_get_last_s (objects_cache_t ∗ocp)

  Gets the least recently used object buffer from the LRU list.
- void chCacheObjectInit (objects_cache_t ∗ocp, ucnt_t hashn, oc_hash_header_t ∗hashp, ucnt_t objn, size↩ _t objsz, void ∗objvp, oc_readf_t readf, oc_writef_t writef)

  Initializes a `objects_cache_t` object.
- oc_object_t ∗ chCacheGetObject (objects_cache_t ∗ocp, uint32_t group, uint32_t key)

  Retrieves an object from the cache.
- void chCacheReleaseObjectI (objects_cache_t ∗ocp, oc_object_t ∗objp)

  Releases an object into the cache.
- bool chCacheReadObject (objects_cache_t ∗ocp, oc_object_t ∗objp, bool async)

  Reads object data from the storage.
- bool chCacheWriteObject (objects_cache_t ∗ocp, oc_object_t ∗objp, bool async)

  Writes the object data back to storage.
- static void chCacheReleaseObject (objects_cache_t ∗ocp, oc_object_t ∗objp)

  Releases an object into the cache.

6.4.5.3.2 Macro Definition Documentation

6.4.5.3.2.1 OC_HASH_FUNCTION

```
#define OC_HASH_FUNCTION(
              ocp,
              group,
              key)
```

Value:

```
(((unsigned)(group) + (unsigned)(key)) & ((unsigned)(ocp)->hashn - 1U))
```

6.4.5.3.2.2 HASH_INSERT

```
#define HASH_INSERT(
                ocp,
                objp,
                group,
                key)
```

Value:

```
  {                                 \
  oc_hash_header_t *hhp;                                                \
  (hhp) = &(ocp)->hashp[OC_HASH_FUNCTION(ocp, group, key)];             \
  (objp)->hash_next = (hhp)->hash_next;                                 \
  (objp)->hash_prev = (oc_object_t *)(hhp);                             \
  (hhp)->hash_next->hash_prev = (objp);                                 \
  (hhp)->hash_next = (objp);                                            \
}
```

6.4.5.3.2.3 HASH_REMOVE

```
#define HASH_REMOVE(
                objp)
```

Value:

```
  {                                                 \
  (objp)->hash_prev->hash_next = (objp)->hash_next;                     \
  (objp)->hash_next->hash_prev = (objp)->hash_prev;                     \
}
```

6.4.5.3.2.4 LRU_INSERT_HEAD

```
#define LRU_INSERT_HEAD(
                ocp,
                objp)
```

Value:

```
  {                                     \
  (objp)->lru_next = (ocp)->lru.lru_next;                               \
  (objp)->lru_prev = (oc_object_t *)&(ocp)->lru;                        \
  (ocp)->lru.lru_next->lru_prev = (objp);                               \
  (ocp)->lru.lru_next = (objp);                                         \
}
```

6.4.5.3.2.5 LRU_INSERT_TAIL

```
#define LRU_INSERT_TAIL(
                ocp,
                objp)
```

Value:

```
  {                                     \
  (objp)->lru_prev = (ocp)->lru.lru_prev;                               \
  (objp)->lru_next = (oc_object_t *)&(ocp)->lru;                        \
  (ocp)->lru.lru_prev->lru_next = (objp);                               \
  (ocp)->lru.lru_prev = (objp);                                         \
}
```

6.4.5.3.2.6 LRU_REMOVE

```
#define LRU_REMOVE(
                objp)
```

Value:

```
  {                                                   \
  (objp)->lru_prev->lru_next = (objp)->lru_next;                          \
  (objp)->lru_next->lru_prev = (objp)->lru_prev;                          \
}
```

6.4.5.3.2.7 OC_FLAG_INLRU

```
#define OC_FLAG_INLRU 0x00000001U
```

6.4.5.3.2.8 OC_FLAG_INHASH

```
#define OC_FLAG_INHASH 0x00000002U
```

6.4.5.3.2.9 OC_FLAG_SHARED

```
#define OC_FLAG_SHARED 0x00000004U
```

6.4.5.3.2.10 OC_FLAG_NOTSYNC

```
#define OC_FLAG_NOTSYNC 0x00000008U
```

6.4.5.3.2.11 OC_FLAG_LAZYWRITE

```
#define OC_FLAG_LAZYWRITE 0x00000010U
```

6.4.5.3.2.12 OC_FLAG_FORGET

```
#define OC_FLAG_FORGET 0x00000020U
```

6.4.5.3.3 Typedef Documentation

6.4.5.3.3.1 oc_flags_t

```
typedef uint32_t oc_flags_t
```

Flags of cached objects.

6.4.5.3.3.2 oc_hash_header_t

```
typedef struct ch_oc_hash_header oc_hash_header_t
```

Type of an hash element header.

6.4.5.3.3.3 oc_lru_header_t

```
typedef struct ch_oc_lru_header oc_lru_header_t
```

Type of an LRU element header.

6.4.5.3.3.4 oc_object_t

```
typedef struct ch_oc_object oc_object_t
```

Type of a cached object.

6.4.5.3.3.5 objects_cache_t

```
typedef struct ch_objects_cache objects_cache_t
```

Type of a cache object.

6.4.5.3.3.6 oc_readf_t

```
typedef bool(* oc_readf_t) (objects_cache_t *ocp, oc_object_t *objp, bool async)
```

Object read function.

Parameters

in	*ocp*	pointer to the objects_cache_t structure
in	*async*	requests an asynchronous operation if supported, the function is then responsible for releasing the object

6.4.5.3.3.7 oc_writef_t

```
typedef bool(* oc_writef_t) (objects_cache_t *ocp, oc_object_t *objp, bool async)
```

Object write function.

Parameters

in	*ocp*	pointer to the objects_cache_t structure
in	*async*	requests an asynchronous operation if supported, the function is then responsible for releasing the object

6.4.5.3.4 Function Documentation

6.4.5.3.4.1 hash_get_s()

```
oc_object_t * hash_get_s (
              objects_cache_t * ocp,
              uint32_t group,
              uint32_t key)  [static]
```

Returns an object pointer from the cache, if present.

Parameters

out	*ocp*	pointer to the objects_cache_t structure to be
in	*group*	object group identifier
in	*key*	object identifier within the group initialized

Returns

The pointer to the retrieved object.

Return values

NULL	if the object is not in cache.

Function Class:

Not an API, this function is for internal use only.

6.4.5.3.4.2 lru_get_last_s()

```
oc_object_t * lru_get_last_s (
              objects_cache_t * ocp)  [static]
```

Gets the least recently used object buffer from the LRU list.

Parameters

out	*ocp*	pointer to the objects_cache_t structure to be

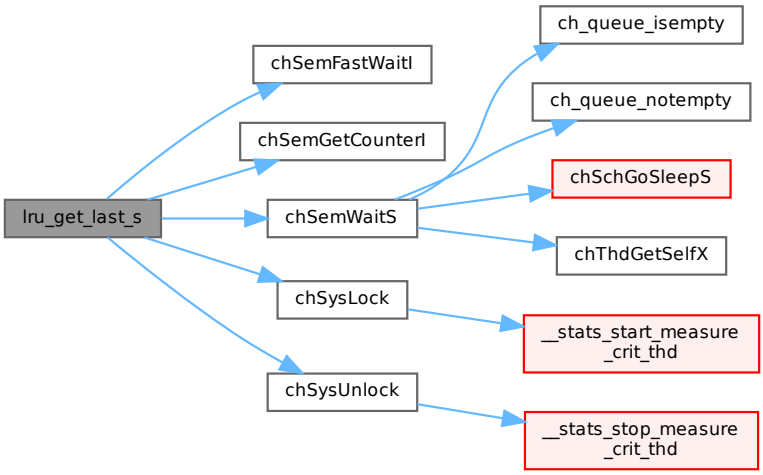
Returns

The pointer to the retrieved object.

Function Class:

Not an API, this function is for internal use only.

Here is the call graph for this function:



6.4.5.3.4.3 chCacheObjectInit()

```
void chCacheObjectInit (
              objects_cache_t * ocp,
              ucnt_t hashn,
              oc_hash_header_t * hashp,
              ucnt_t objn,
              size_t objsz,
              void * objvp,
              oc_readf_t readf,
              oc_writef_t writef)
```

Initializes a `objects_cache_t` object.

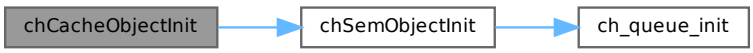
Parameters

out	*ocp*	pointer to the `objects_cache_t` structure to be initialized
in	*hashn*	number of elements in the hash table array, must be a power of two and not lower than `objn`
in	*hashp*	pointer to the hash table as an array of `oc_hash_header_t`
in	*objn*	number of elements in the objects table array
in	*objsz*	size of elements in the objects table array, the minimum value is `sizeof (oc_object_t)`.
in	*objvp*	pointer to the hash objects as an array of structures starting with an `oc_object_t`
in	*readf*	pointer to an object reader function
in	*writef*	pointer to an object writer function

Function Class:

Object or module nitializer function.

Here is the call graph for this function:



6.4.5.3.4.4 chCacheGetObject()

```
oc_object_t * chCacheGetObject (
              objects_cache_t * ocp,
              uint32_t group,
              uint32_t key)
```

Retrieves an object from the cache.

Note

If the object is not in cache then the returned object is marked as OC_FLAG_NOTSYNC meaning that its data contains garbage and must be initialized.

Parameters

in	*ocp*	pointer to the objects_cache_t structure
in	*group*	object group identifier
in	*key*	object identifier within the group

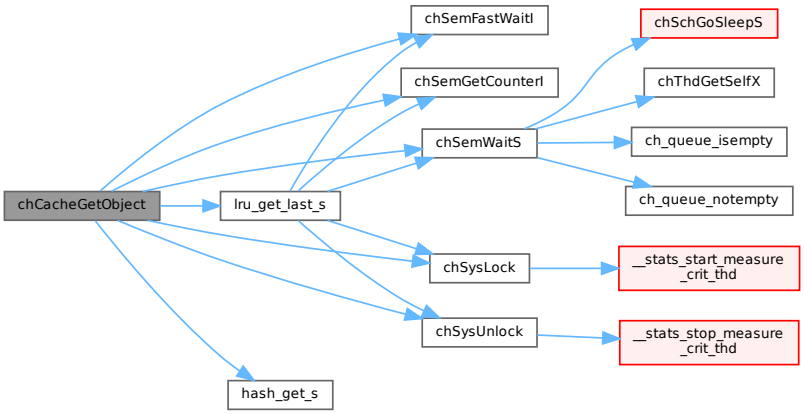
Returns

The pointer to the retrieved object.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



6.4.5.3.4.5 chCacheReleaseObjectI()

```
void chCacheReleaseObjectI (
             objects_cache_t * ocp,
             oc_object_t * objp)
```

Releases an object into the cache.

Note

This function gives a meaning to the following flags:

- `OC_FLAG_INLRU` must be cleared.
- `OC_FLAG_INHASH` must be set.
- `OC_FLAG_SHARED` must be cleared.
- `OC_FLAG_NOTSYNC` invalidates the object and queues it on the LRU tail.
- `OC_FLAG_LAZYWRITE` is ignored and kept, a write will occur when the object is removed from the LRU list (lazy write).

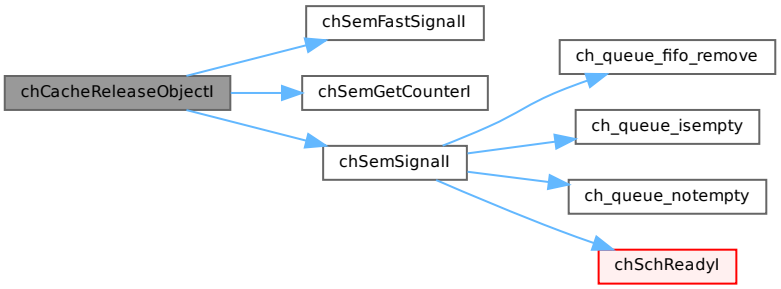
Parameters

in	*ocp*	pointer to the `objects_cache_t` structure
in	*objp*	pointer to the `oc_object_t` structure

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:



6.4.5.3.4.6 chCacheReadObject()

```
bool chCacheReadObject (
              objects_cache_t * ocp,
              oc_object_t * objp,
              bool async)
```

Reads object data from the storage.

Note

In case of asynchronous operation an error condition is not reported by this function.

Parameters

in	*ocp*	pointer to the objects_cache_t structure
in	*objp*	pointer to the oc_object_t structure
in	*async*	requests an asynchronous operation if supported, the function is then responsible for releasing the object

Returns

The operation status. In case of asynchronous operation false is always returned.

Return values

false	if the operation succeeded.
true	if the synchronous read operation failed.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

6.4.5.3.4.7 chCacheWriteObject()

```
bool chCacheWriteObject (
             objects_cache_t * ocp,
             oc_object_t * objp,
             bool async)
```

Writes the object data back to storage.

Note

In case of asynchronous operation an error condition is not reported by this function.

Parameters

in	*ocp*	pointer to the objects_cache_t structure
in	*objp*	pointer to the oc_object_t structure
in	*async*	requests an asynchronous operation if supported, the function is then responsible for releasing the object

Returns

The operation status. In case of asynchronous operation false is always returned.

Return values

false	if the operation succeeded.
true	if the synchronous write operation failed.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

6.4.5.3.4.8 chCacheReleaseObject()

```
void chCacheReleaseObject (
             objects_cache_t * ocp,
             oc_object_t * objp)  [inline], [static]
```

Releases an object into the cache.

Note

This function gives a meaning to the following flags:

- OC_FLAG_INLRU must be cleared.
- OC_FLAG_INHASH must be set.
- OC_FLAG_SHARED must be cleared.
- OC_FLAG_NOTSYNC invalidates the object and queues it on the LRU tail.
- OC_FLAG_LAZYWRITE is ignored and kept, a write will occur when the object is removed from the LRU list (lazy write).

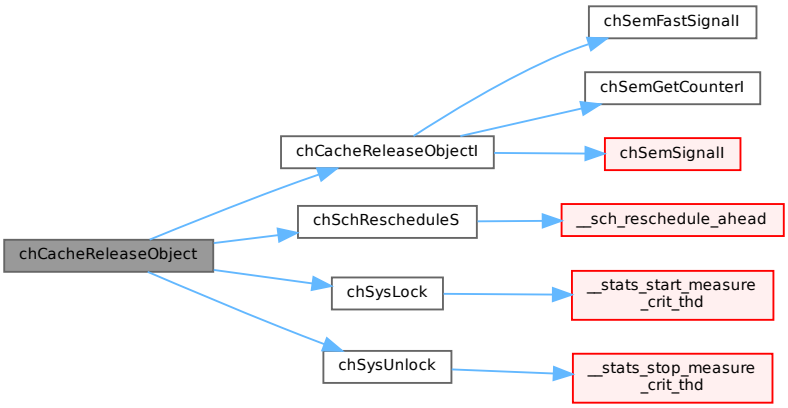
Parameters

in	*ocp*	pointer to the objects_cache_t structure
in	*objp*	pointer to the oc_object_t structure

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



6.4.5.4 Dynamic Objects Factory

6.4.5.4.1 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.

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 F_LOCK()
- #define F_UNLOCK()
- #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 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.
- #define CH_FACTORY_REQUIRES_POOLS
- #define CH_FACTORY_REQUIRES_HEAP

Typedefs

- typedef struct ch_dyn_element dyn_element_t

  Type of a dynamic object list element.
- typedef struct ch_dyn_list dyn_list_t

  Type of a dynamic object list.
- typedef struct ch_registered_static_object registered_object_t

  Type of a registered object.
- typedef struct ch_dyn_object dyn_buffer_t

  Type of a dynamic buffer object.
- typedef struct ch_dyn_semaphore dyn_semaphore_t

  Type of a dynamic semaphore.
- typedef struct ch_dyn_mailbox dyn_mailbox_t

  Type of a dynamic buffer object.
- typedef struct ch_dyn_objects_fifo dyn_objects_fifo_t

  Type of a dynamic buffer object.
- typedef struct ch_dyn_pipe dyn_pipe_t

  Type of a dynamic pipe object.
- typedef struct ch_objects_factory objects_factory_t

  Type of the factory main object.

Functions

- static void copy_name (const char *sp, char *dp)
- static void dyn_list_init (dyn_list_t *dlp)
- static dyn_element_t * dyn_list_find (const char *name, dyn_list_t *dlp)
- static dyn_element_t * dyn_list_find_prev (dyn_element_t *element, dyn_list_t *dlp)
- static dyn_element_t * dyn_list_unlink (dyn_element_t *prev)
- static dyn_element_t * dyn_create_object_heap (const char *name, dyn_list_t *dlp, size_t size, unsigned align)
- static void dyn_release_object_heap (dyn_element_t *dep, dyn_list_t *dlp)
- static dyn_element_t * dyn_create_object_pool (const char *name, dyn_list_t *dlp, memory_pool_t *mp)
- static void dyn_release_object_pool (dyn_element_t *dep, dyn_list_t *dlp, memory_pool_t *mp)
- static dyn_element_t * dyn_find_object (const char *name, dyn_list_t *dlp)
- void __factory_init (void)

  Initializes the objects factory.
- registered_object_t * chFactoryRegisterObject (const char *name, void *objp)

  Registers a generic object.
- registered_object_t * chFactoryFindObject (const char *name)

  Retrieves a registered object.
- registered_object_t * chFactoryFindObjectByPointer (void *objp)

  Retrieves a registered object by pointer.
- void chFactoryReleaseObject (registered_object_t *rop)

  Releases a registered object.
- dyn_buffer_t * chFactoryCreateBuffer (const char *name, size_t size)

  Creates a generic dynamic buffer object.
- dyn_buffer_t * chFactoryFindBuffer (const char *name)

  Retrieves a dynamic buffer object.
- void chFactoryReleaseBuffer (dyn_buffer_t *dbp)

  Releases a dynamic buffer object.
- dyn_semaphore_t * chFactoryCreateSemaphore (const char *name, cnt_t n)

Creates a dynamic semaphore object.

- dyn_semaphore_t * chFactoryFindSemaphore (const char *name)

  Retrieves a dynamic semaphore object.
- void chFactoryReleaseSemaphore (dyn_semaphore_t *dsp)

  Releases a dynamic semaphore object.
- dyn_mailbox_t * chFactoryCreateMailbox (const char *name, size_t n)

  Creates a dynamic mailbox object.
- dyn_mailbox_t * chFactoryFindMailbox (const char *name)

  Retrieves a dynamic mailbox object.
- void chFactoryReleaseMailbox (dyn_mailbox_t *dmp)

  Releases a dynamic mailbox object.
- dyn_objects_fifo_t * chFactoryCreateObjectsFIFO (const char *name, size_t objsize, size_t objn, unsigned objalign)

  Creates a dynamic "objects FIFO" object.
- dyn_objects_fifo_t * chFactoryFindObjectsFIFO (const char *name)

  Retrieves a dynamic "objects FIFO" object.
- void chFactoryReleaseObjectsFIFO (dyn_objects_fifo_t *dofp)

  Releases a dynamic "objects FIFO" object.
- dyn_pipe_t * chFactoryCreatePipe (const char *name, size_t size)

  Creates a dynamic pipe object.
- dyn_pipe_t * chFactoryFindPipe (const char *name)

  Retrieves a dynamic pipe object.
- void chFactoryReleasePipe (dyn_pipe_t *dpp)

  Releases a dynamic pipe object.
- static dyn_element_t * chFactoryDuplicateReference (dyn_element_t *dep)

  Duplicates an object reference.
- static void * chFactoryGetObject (registered_object_t *rop)

  Returns the pointer to the inner registered object.
- static size_t chFactoryGetBufferSize (dyn_buffer_t *dbp)

  Returns the size of a generic dynamic buffer object.
- static uint8_t * chFactoryGetBuffer (dyn_buffer_t *dbp)

  Returns the pointer to the inner buffer.
- static semaphore_t * chFactoryGetSemaphore (dyn_semaphore_t *dsp)

  Returns the pointer to the inner semaphore.
- static mailbox_t * chFactoryGetMailbox (dyn_mailbox_t *dmp)

  Returns the pointer to the inner mailbox.
- static objects_fifo_t * chFactoryGetObjectsFIFO (dyn_objects_fifo_t *dofp)

  Returns the pointer to the inner objects FIFO.
- static pipe_t * chFactoryGetPipe (dyn_pipe_t *dpp)

  Returns the pointer to the inner pipe.

Variables

- objects_factory_t ch_factory

  Factory object static instance.

6.4.5.4.2 Macro Definition Documentation

6.4.5.4.2.1 F_LOCK

```
#define F_LOCK()
```

Value:

```
chMtxLock(&ch_factory.mtx)
```

6.4.5.4.2.2 F_UNLOCK

```
#define F_UNLOCK()
```

Value:

```
chMtxUnlock(&ch_factory.mtx)
```

6.4.5.4.2.3 CH_CFG_FACTORY_MAX_NAMES_LENGTH

```
#define CH_CFG_FACTORY_MAX_NAMES_LENGTH 8
```

Maximum length for object names.

If the specified length is zero then the name is stored by pointer but this could have unintended side effects.

6.4.5.4.2.4 CH_CFG_FACTORY_OBJECTS_REGISTRY

```
#define CH_CFG_FACTORY_OBJECTS_REGISTRY TRUE
```

Enables the registry of generic objects.

6.4.5.4.2.5 CH_CFG_FACTORY_GENERIC_BUFFERS

```
#define CH_CFG_FACTORY_GENERIC_BUFFERS TRUE
```

Enables factory for generic buffers.

6.4.5.4.2.6 CH_CFG_FACTORY_SEMAPHORES [1/2]

```
#define CH_CFG_FACTORY_SEMAPHORES TRUE
```

Enables factory for semaphores.

6.4.5.4.2.7 CH_CFG_FACTORY_SEMAPHORES [2/2]

```
#define CH_CFG_FACTORY_SEMAPHORES FALSE
```

Enables factory for semaphores.

6.4.5.4.2.8 CH_CFG_FACTORY_MAILBOXES [1/2]

```
#define CH_CFG_FACTORY_MAILBOXES TRUE
```

Enables factory for mailboxes.

6.4.5.4.2.9 CH_CFG_FACTORY_MAILBOXES [2/2]

```
#define CH_CFG_FACTORY_MAILBOXES FALSE
```

Enables factory for mailboxes.

6.4.5.4.2.10 CH_CFG_FACTORY_OBJ_FIFOS [1/3]

```
#define CH_CFG_FACTORY_OBJ_FIFOS TRUE
```

Enables factory for objects FIFOs.

6.4.5.4.2.11 CH_CFG_FACTORY_OBJ_FIFOS [2/3]

```
#define CH_CFG_FACTORY_OBJ_FIFOS TRUE
```

Enables factory for objects FIFOs.

6.4.5.4.2.12 CH_CFG_FACTORY_OBJ_FIFOS [3/3]

```
#define CH_CFG_FACTORY_OBJ_FIFOS FALSE
```

Enables factory for objects FIFOs.

6.4.5.4.2.13 CH_CFG_FACTORY_PIPES [1/2]

```
#define CH_CFG_FACTORY_PIPES TRUE
```

Enables factory for Pipes.

6.4.5.4.2.14 CH_CFG_FACTORY_PIPES [2/2]

```
#define CH_CFG_FACTORY_PIPES FALSE
```

Enables factory for Pipes.

6.4.5.4.2.15 CH_FACTORY_REQUIRES_POOLS

```
#define CH_FACTORY_REQUIRES_POOLS
```

Value:

```
  ((CH_CFG_FACTORY_OBJECTS_REGISTRY == TRUE) ||                         \
   (CH_CFG_FACTORY_SEMAPHORES == TRUE))
```

6.4.5.4.2.16 CH_FACTORY_REQUIRES_HEAP

```
#define CH_FACTORY_REQUIRES_HEAP
```

Value:

```
  ((CH_CFG_FACTORY_GENERIC_BUFFERS == TRUE) ||                          \
   (CH_CFG_FACTORY_MAILBOXES == TRUE) ||                                \
   (CH_CFG_FACTORY_OBJ_FIFOS == TRUE) ||                                \
   (CH_CFG_FACTORY_PIPES == TRUE))
```

6.4.5.4.3 Typedef Documentation

6.4.5.4.3.1 dyn_element_t

```
typedef struct ch_dyn_element dyn_element_t
```

Type of a dynamic object list element.

6.4.5.4.3.2 dyn_list_t

```
typedef struct ch_dyn_list dyn_list_t
```

Type of a dynamic object list.

6.4.5.4.3.3 registered_object_t

```
typedef struct ch_registered_static_object registered_object_t
```

Type of a registered object.

6.4.5.4.3.4 dyn_buffer_t

```
typedef struct ch_dyn_object dyn_buffer_t
```

Type of a dynamic buffer object.

6.4.5.4.3.5 dyn_semaphore_t

```
typedef struct ch_dyn_semaphore dyn_semaphore_t
```

Type of a dynamic semaphore.

6.4.5.4.3.6 dyn_mailbox_t

```
typedef struct ch_dyn_mailbox dyn_mailbox_t
```

Type of a dynamic buffer object.

6.4.5.4.3.7 dyn_objects_fifo_t

```
typedef struct ch_dyn_objects_fifo dyn_objects_fifo_t
```

Type of a dynamic buffer object.

6.4.5.4.3.8 dyn_pipe_t

```
typedef struct ch_dyn_pipe dyn_pipe_t
```

Type of a dynamic pipe object.

6.4.5.4.3.9 objects_factory_t

```
typedef struct ch_objects_factory objects_factory_t
```

Type of the factory main object.

6.4.5.4.4 Function Documentation

6.4.5.4.4.1 copy_name()

```
void copy_name (
            const char * sp,
            char * dp)  [static]
```

6.4.5.4.4.2 dyn_list_init()

```
void dyn_list_init (
            dyn_list_t * dlp)  [inline], [static]
```

6.4.5.4.4.3 dyn_list_find()

```
dyn_element_t * dyn_list_find (
            const char * name,
            dyn_list_t * dlp)  [static]
```

6.4.5.4.4.4 dyn_list_find_prev()

```
dyn_element_t * dyn_list_find_prev (
            dyn_element_t * element,
            dyn_list_t * dlp)  [static]
```

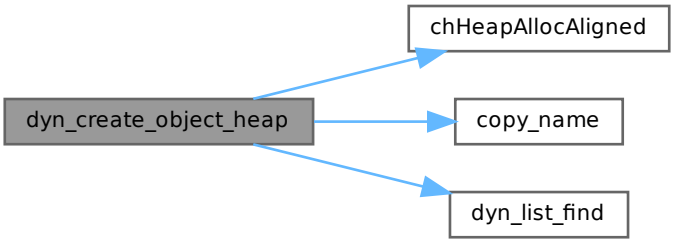
6.4.5.4.4.5 dyn_list_unlink()

```
dyn_element_t * dyn_list_unlink (
            dyn_element_t * prev)  [static]
```

6.4.5.4.4.6 dyn_create_object_heap()

```
dyn_element_t * dyn_create_object_heap (
            const char * name,
            dyn_list_t * dlp,
            size_t size,
            unsigned align)  [static]
```

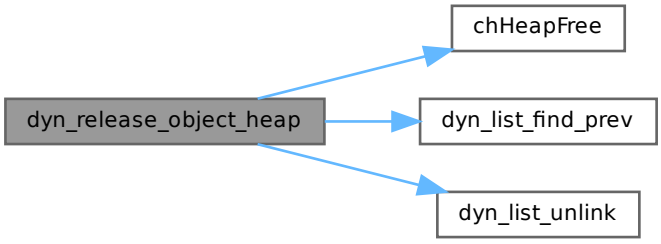
Here is the call graph for this function:



6.4.5.4.4.7 dyn_release_object_heap()

```
void dyn_release_object_heap (
            dyn_element_t * dep,
            dyn_list_t * dlp)  [static]
```

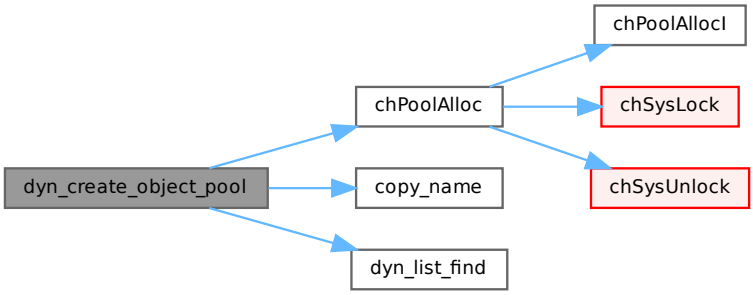
Here is the call graph for this function:



6.4.5.4.4.8 dyn_create_object_pool()

```
dyn_element_t * dyn_create_object_pool (
            const char * name,
            dyn_list_t * dlp,
            memory_pool_t * mp)  [static]
```

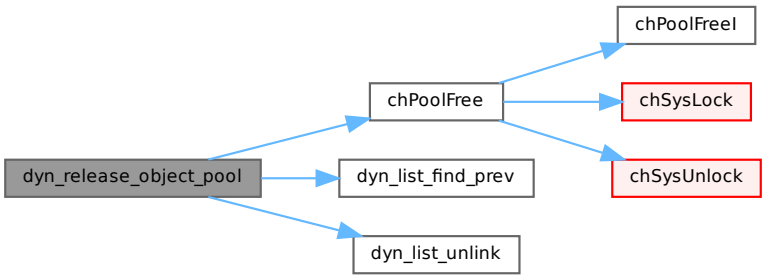
Here is the call graph for this function:



6.4.5.4.4.9 dyn_release_object_pool()

```
void dyn_release_object_pool (
            dyn_element_t * dep,
            dyn_list_t * dlp,
            memory_pool_t * mp)  [static]
```

Here is the call graph for this function:



6.4.5.4.4.10 dyn_find_object()

```
dyn_element_t * dyn_find_object (
            const char * name,
            dyn_list_t * dlp)  [static]
```

Here is the call graph for this function:

6.4.5.4.4.11 __factory_init()

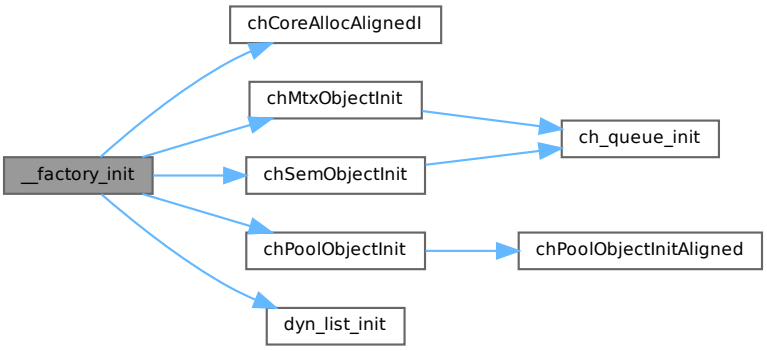
```
void __factory_init (
            void )
```

Initializes the objects factory.

Function Class:

Object or module nitializer function.

Here is the call graph for this function:



6.4.5.4.4.12 chFactoryRegisterObject()

```
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

in	*name*	name to be assigned to the registered object
in	*objp*	pointer to the object to be registered

Returns

The reference to the registered object.

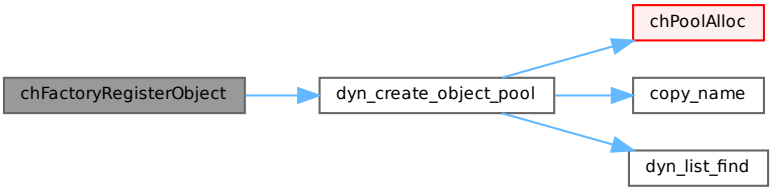
Return values

NULL	if the object to be registered cannot be allocated or a registered object with the same name exists.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



6.4.5.4.4.13 chFactoryFindObject()

```
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.

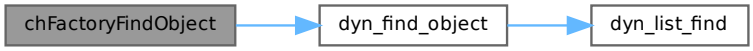
Return values

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.

Here is the call graph for this function:



6.4.5.4.4.14 chFactoryFindObjectByPointer()

```
registered_object_t * chFactoryFindObjectByPointer (
            void * objp)
```

Retrieves a registered object by pointer.

Postcondition

A reference to the registered object is returned with the reference counter increased by one.

Parameters

in	*objp*	pointer to the object to be retrieved

Returns

The reference to the found registered object.

Return values

NULL	if a registered object with the specified pointer does not exist.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

6.4.5.4.4.15 chFactoryReleaseObject()

```
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.

Here is the call graph for this function:

6.4.5.4.4.16 chFactoryCreateBuffer()

```
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.

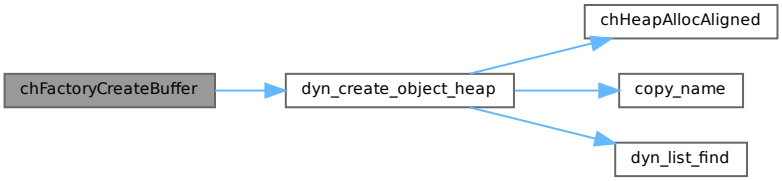
Return values

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.

Here is the call graph for this function:



6.4.5.4.4.17 chFactoryFindBuffer()

```
dyn_buffer_t * chFactoryFindBuffer (
            const char * name)
```

Retrieves a dynamic buffer object.

Postcondition

A reference to the dynamic buffer object is returned with the reference counter increased by one.

Parameters

in	*name*	name of the dynamic buffer object

Returns

The reference to the found dynamic buffer object.

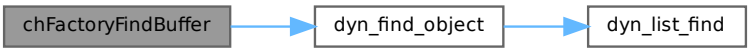
Return values

NULL	if a dynamic buffer object with the specified name does not exist.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



6.4.5.4.4.18 chFactoryReleaseBuffer()

```
void chFactoryReleaseBuffer (
             dyn_buffer_t * dbp)
```

Releases a dynamic buffer object.

The reference counter of the dynamic buffer object is decreased by one, if reaches zero then the dynamic buffer object memory is freed.

Parameters

in	*dbp*	dynamic buffer object reference

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

6.4.5.4.4.19 chFactoryCreateSemaphore()

```
dyn_semaphore_t * chFactoryCreateSemaphore (
             const char * name,
             cnt_t n)
```

Creates a dynamic semaphore object.

Postcondition

A reference to the dynamic semaphore object is returned and the reference counter is initialized to one.

The dynamic semaphore object is initialized and ready to use.

Parameters

in	*name*	name to be assigned to the new dynamic semaphore object
in	*n*	dynamic semaphore object counter initialization value

Returns

The reference to the created dynamic semaphore object.

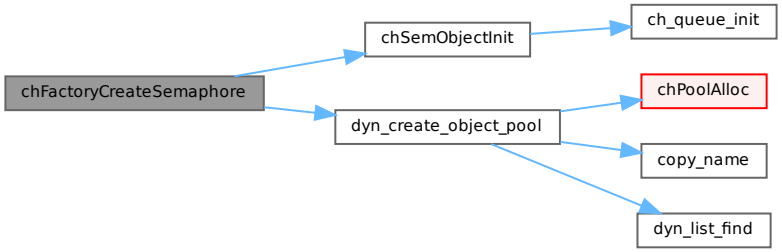
Return values

NULL	if the dynamic semaphore object cannot be allocated or a dynamic semaphore with the same name exists.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



6.4.5.4.4.20 chFactoryFindSemaphore()

```
dyn_semaphore_t * chFactoryFindSemaphore (
            const char * name)
```

Retrieves a dynamic semaphore object.

Postcondition

A reference to the dynamic semaphore object is returned with the reference counter increased by one.

Parameters

in	*name*	name of the dynamic semaphore object

Returns

The reference to the found dynamic semaphore object.

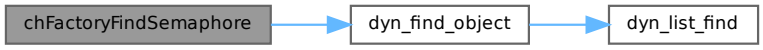
Return values

NULL	if a dynamic semaphore object with the specified name does not exist.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



6.4.5.4.4.21 chFactoryReleaseSemaphore()

```
void chFactoryReleaseSemaphore (
            dyn_semaphore_t * dsp)
```

Releases a dynamic semaphore object.

The reference counter of the dynamic semaphore object is decreased by one, if reaches zero then the dynamic semaphore object memory is freed.

Parameters

in	*dsp*	dynamic semaphore object reference

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

6.4.5.4.4.22 chFactoryCreateMailbox()

```
dyn_mailbox_t * chFactoryCreateMailbox (
            const char * name,
            size_t n)
```

Creates a dynamic mailbox object.

Postcondition

A reference to the dynamic mailbox object is returned and the reference counter is initialized to one.

The dynamic mailbox object is initialized and ready to use.

Parameters

in	*name*	name to be assigned to the new dynamic mailbox object
in	*n*	mailbox buffer size as number of messages

Returns

The reference to the created dynamic mailbox object.

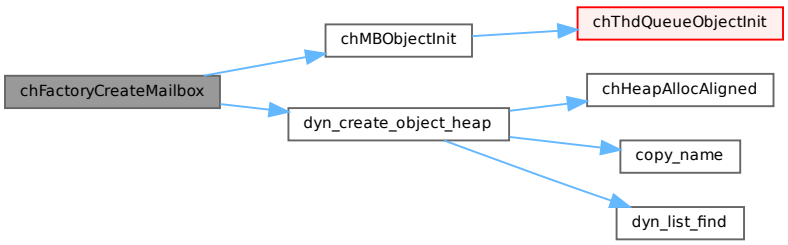
Return values

NULL	if the dynamic mailbox object cannot be allocated or a dynamic mailbox object with the same name exists.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



6.4.5.4.4.23 chFactoryFindMailbox()

```
dyn_mailbox_t * chFactoryFindMailbox (
            const char * name)
```

Retrieves a dynamic mailbox object.

Postcondition

A reference to the dynamic mailbox object is returned with the reference counter increased by one.

Parameters

in	*name*	name of the dynamic mailbox object

Returns

The reference to the found dynamic mailbox object.

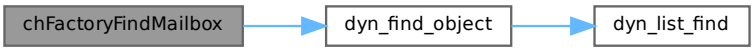
Return values

NULL	if a dynamic mailbox object with the specified name does not exist.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



6.4.5.4.4.24 chFactoryReleaseMailbox()

```
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.

Here is the call graph for this function:

6.4.5.4.4.25 chFactoryCreateObjectsFIFO()

```
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.

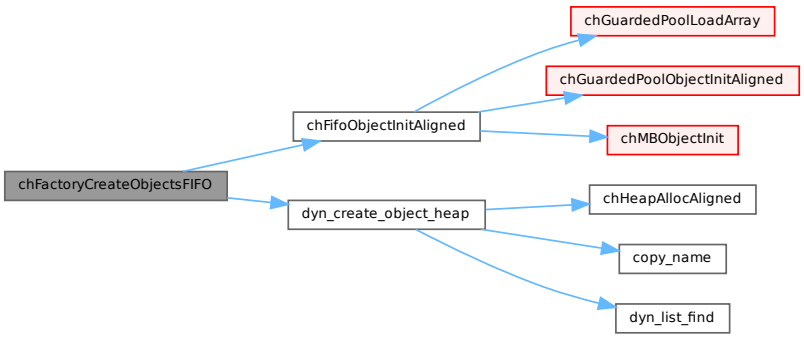
Return values

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:



6.4.5.4.4.26 chFactoryFindObjectsFIFO()

```
dyn_objects_fifo_t * chFactoryFindObjectsFIFO (
            const char * name)
```

Retrieves a dynamic "objects FIFO" object.

Postcondition

A reference to the dynamic "objects FIFO" object is returned with the reference counter increased by one.

Parameters

in	*name*	name of the dynamic "objects FIFO" object

Returns

The reference to the found dynamic "objects FIFO" object.

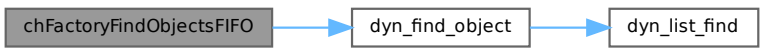
Return values

NULL	if a dynamic "objects FIFO" object with the specified name does not exist.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



6.4.5.4.4.27 chFactoryReleaseObjectsFIFO()

```
void chFactoryReleaseObjectsFIFO (
            dyn_objects_fifo_t * dofp)
```

Releases a dynamic "objects FIFO" object.

The reference counter of the dynamic "objects FIFO" object is decreased by one, if reaches zero then the dynamic "objects FIFO" object memory is freed.

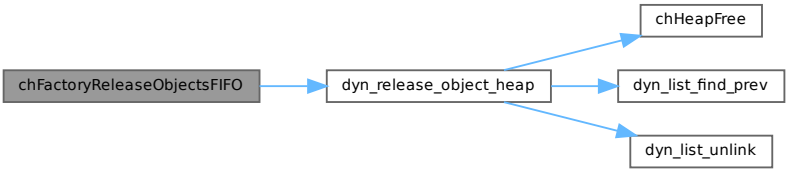
Parameters

in	*dofp*	dynamic "objects FIFO" object reference

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



6.4.5.4.4.28 chFactoryCreatePipe()

```
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.

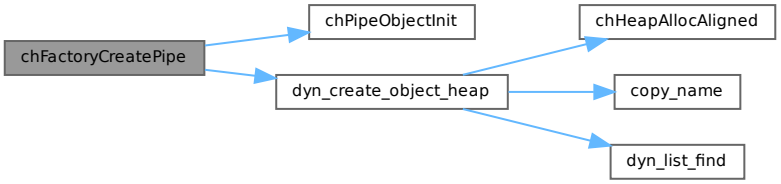
Return values

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:



6.4.5.4.4.29 chFactoryFindPipe()

```
dyn_pipe_t * chFactoryFindPipe (
             const char * name)
```

Retrieves a dynamic pipe object.

Postcondition

A reference to the dynamic pipe object is returned with the reference counter increased by one.

Parameters

in	*name*	name of the pipe object

Returns

The reference to the found dynamic pipe object.

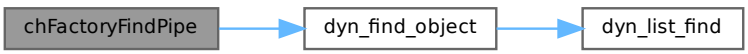
Return values

NULL	if a dynamic pipe object with the specified name does not exist.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



6.4.5.4.4.30 chFactoryReleasePipe()

```
void chFactoryReleasePipe (
            dyn_pipe_t * dpp)
```

Releases a dynamic pipe object.

The reference counter of the dynamic pipe object is decreased by one, if reaches zero then the dynamic pipe object memory is freed.

Parameters

in	*dpp*	dynamic pipe object reference

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

6.4.5.4.4.31 chFactoryDuplicateReference()

```
dyn_element_t * chFactoryDuplicateReference (
            dyn_element_t * dep)  [inline], [static]
```

Duplicates an object reference.

Note

This function can be used on any kind of dynamic object.

Parameters

in	*dep*	pointer to the element field of the object

Returns

The duplicated object reference.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

6.4.5.4.4.32 chFactoryGetObject()

```
void * chFactoryGetObject (
            registered_object_t * rop)  [inline], [static]
```

Returns the pointer to the inner registered object.

Parameters

in	*rop*	registered object reference

Returns

The pointer to the registered object.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

6.4.5.4.4.33 chFactoryGetBufferSize()

```
size_t chFactoryGetBufferSize (
            dyn_buffer_t * dbp)  [inline], [static]
```

Returns the size of a generic dynamic buffer object.

Parameters

in	*dbp*	dynamic buffer object reference

Returns

The size of the buffer object in bytes.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:

6.4.5.4.4.34 chFactoryGetBuffer()

```
uint8_t * chFactoryGetBuffer (
            dyn_buffer_t * dbp)  [inline], [static]
```

Returns the pointer to the inner buffer.

Parameters

in	*dbp*	dynamic buffer object reference

Returns

The pointer to the dynamic buffer.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

6.4.5.4.4.35 chFactoryGetSemaphore()

```
semaphore_t * chFactoryGetSemaphore (
              dyn_semaphore_t * dsp)  [inline], [static]
```

Returns the pointer to the inner semaphore.

Parameters

in	*dsp*	dynamic semaphore object reference

Returns

The pointer to the semaphore.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

6.4.5.4.4.36 chFactoryGetMailbox()

```
mailbox_t * chFactoryGetMailbox (
              dyn_mailbox_t * dmp)  [inline], [static]
```

Returns the pointer to the inner mailbox.

Parameters

in	*dmp*	dynamic mailbox object reference

Returns

The pointer to the mailbox.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

6.4.5.4.4.37 chFactoryGetObjectsFIFO()

```
objects_fifo_t * chFactoryGetObjectsFIFO (
              dyn_objects_fifo_t * dofp)  [inline], [static]
```

Returns the pointer to the inner objects FIFO.

Parameters

in	*dofp*	dynamic "objects FIFO" object reference

Returns

The pointer to the objects FIFO.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

6.4.5.4.4.38 chFactoryGetPipe()

```
pipe_t * chFactoryGetPipe (
             dyn_pipe_t * dpp)  [inline], [static]
```

Returns the pointer to the inner pipe.

Parameters

in	*dpp*	dynamic pipe object reference

Returns

The pointer to the pipe.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

6.4.5.4.5 Variable Documentation

6.4.5.4.5.1 ch_factory

```
objects_factory_t ch_factory
```

Factory object static instance.

Note

It is a global object because it could be accessed through a specific debugger plugin.

Chapter 7

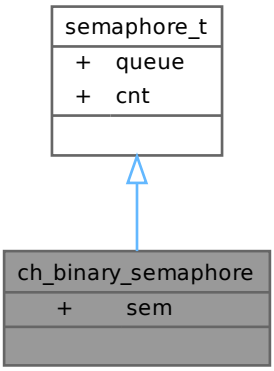
Data Structure Documentation

7.1 ch_binary_semaphore Struct Reference

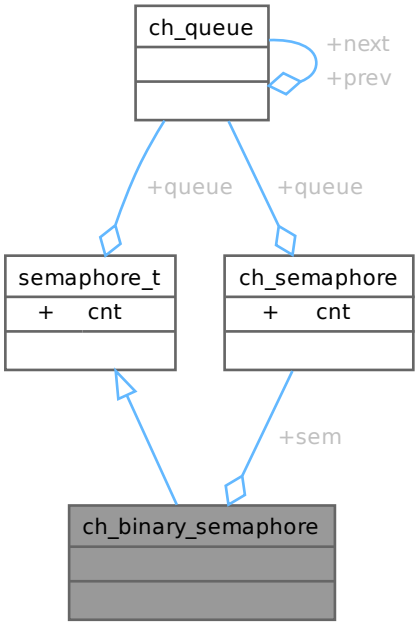
Binary semaphore type.

`#include <chbsem.h>`

Inheritance diagram for ch_binary_semaphore:



Collaboration diagram for ch_binary_semaphore:



Data Fields

- semaphore_t sem

Data Fields inherited from ch_semaphore

- ch_queue_t queue

  Queue of the threads sleeping on this semaphore.
- cnt_t cnt

  The semaphore counter.

7.1.1 Detailed Description

Binary semaphore type.

7.1.2 Field Documentation

7.1.2.1 sem

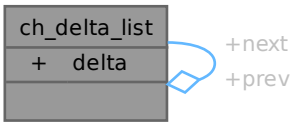
```
semaphore_t ch_binary_semaphore::sem
```

7.2 ch_delta_list Struct Reference

Delta list element and header structure.

`#include <chlists.h>`

Collaboration diagram for ch_delta_list:



Data Fields

- ch_delta_list_t * next
  Next in the delta list.
- ch_delta_list_t * prev
  Previous in the delta list.
- sysinterval_t delta
  Time interval from previous.

7.2.1 Detailed Description

Delta list element and header structure.

7.2.2 Field Documentation

7.2.2.1 next

`ch_delta_list_t* ch_delta_list::next`

Next in the delta list.

7.2.2.2 prev

`ch_delta_list_t* ch_delta_list::prev`

Previous in the delta list.

7.2.2.3 delta

```
sysinterval_t ch_delta_list::delta
```

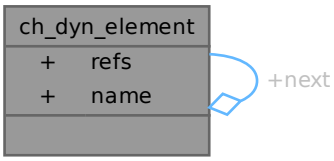
Time interval from previous.

7.3 ch_dyn_element Struct Reference

Type of a dynamic object list element.

```
#include <chfactory.h>
```

Collaboration diagram for ch_dyn_element:



Data Fields

- struct ch_dyn_element * next
  Next dynamic object in the list.
- ucnt_t refs
  Number of references to this object.
- char name [CH_CFG_FACTORY_MAX_NAMES_LENGTH]

7.3.1 Detailed Description

Type of a dynamic object list element.

7.3.2 Field Documentation

7.3.2.1 next

```
struct ch_dyn_element* ch_dyn_element::next
```

Next dynamic object in the list.

7.3.2.2 refs

```
ucnt_t ch_dyn_element::refs
```

Number of references to this object.

7.3.2.3 name

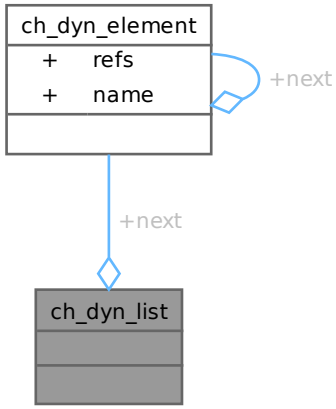
```
char ch_dyn_element::name[CH_CFG_FACTORY_MAX_NAMES_LENGTH]
```

7.4 ch_dyn_list Struct Reference

Type of a dynamic object list.

```
#include <chfactory.h>
```

Collaboration diagram for ch_dyn_list:



Data Fields

- dyn_element_t * next

7.4.1 Detailed Description

Type of a dynamic object list.

7.4.2 Field Documentation

7.4.2.1 next

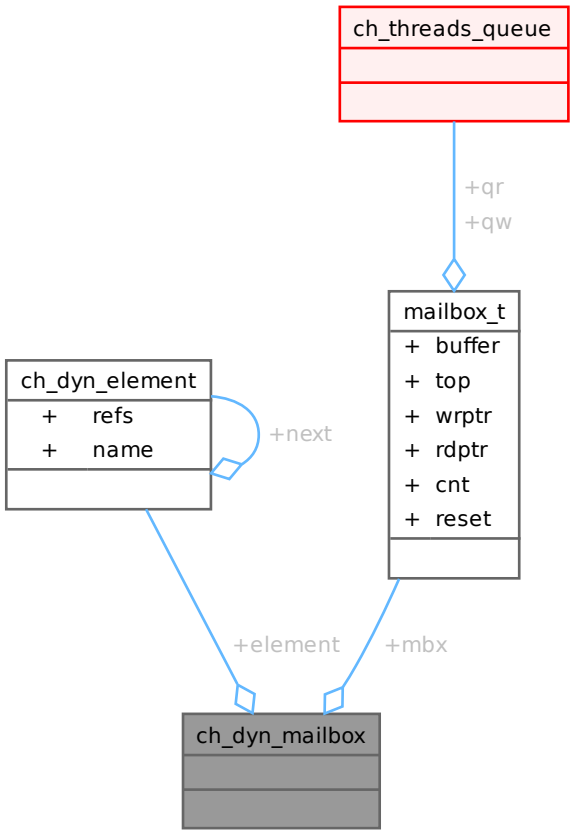
```
dyn_element_t* ch_dyn_list::next
```

7.5 ch_dyn_mailbox Struct Reference

Type of a dynamic buffer object.

```
#include <chfactory.h>
```

Collaboration diagram for ch_dyn_mailbox:



Data Fields

- dyn_element_t element

  List element of the dynamic buffer object.
- mailbox_t mbx

  The mailbox.

7.5.1 Detailed Description

Type of a dynamic buffer object.

7.5.2 Field Documentation

7.5.2.1 element

```
dyn_element_t ch_dyn_mailbox::element
```

List element of the dynamic buffer object.

7.5.2.2 mbx

```
mailbox_t ch_dyn_mailbox::mbx
```

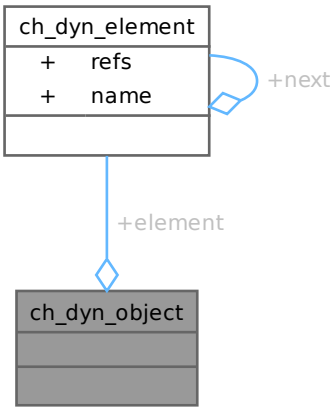
The mailbox.

7.6 ch_dyn_object Struct Reference

Type of a dynamic buffer object.

```
#include <chfactory.h>
```

Collaboration diagram for ch_dyn_object:



Data Fields

- dyn_element_t element

  List element of the dynamic buffer object.

7.6.1 Detailed Description

Type of a dynamic buffer object.

7.6.2 Field Documentation

7.6.2.1 element

```
dyn_element_t ch_dyn_object::element
```

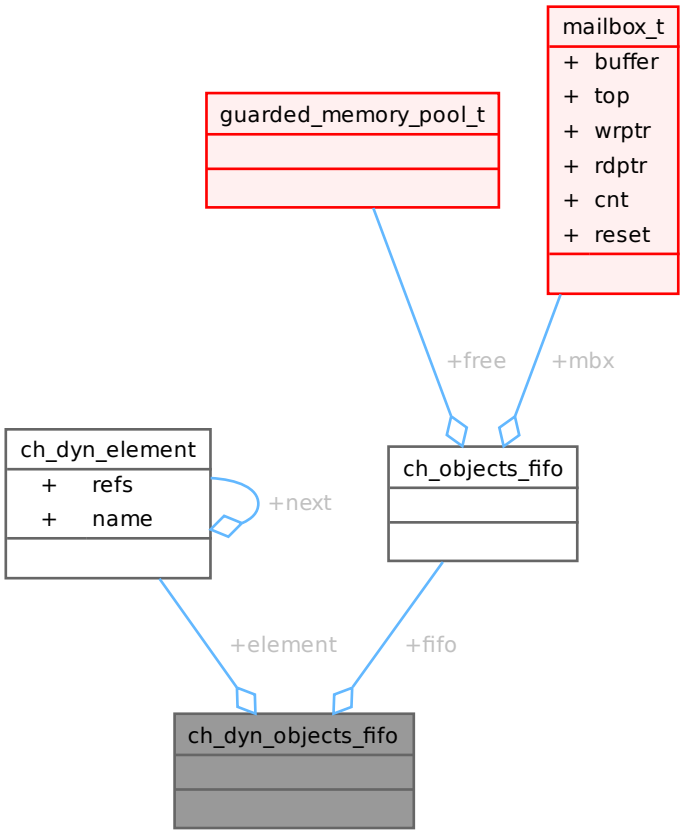
List element of the dynamic buffer object.

7.7 ch_dyn_objects_fifo Struct Reference

Type of a dynamic buffer object.

```
#include <chfactory.h>
```

Collaboration diagram for ch_dyn_objects_fifo:



Data Fields

- dyn_element_t element

  List element of the dynamic buffer object.
- objects_fifo_t fifo

  The objects FIFO.

7.7.1 Detailed Description

Type of a dynamic buffer object.

7.7.2 Field Documentation

7.7.2.1 element

```
dyn_element_t ch_dyn_objects_fifo::element
```

List element of the dynamic buffer object.

7.7.2.2 fifo

```
objects_fifo_t ch_dyn_objects_fifo::fifo
```

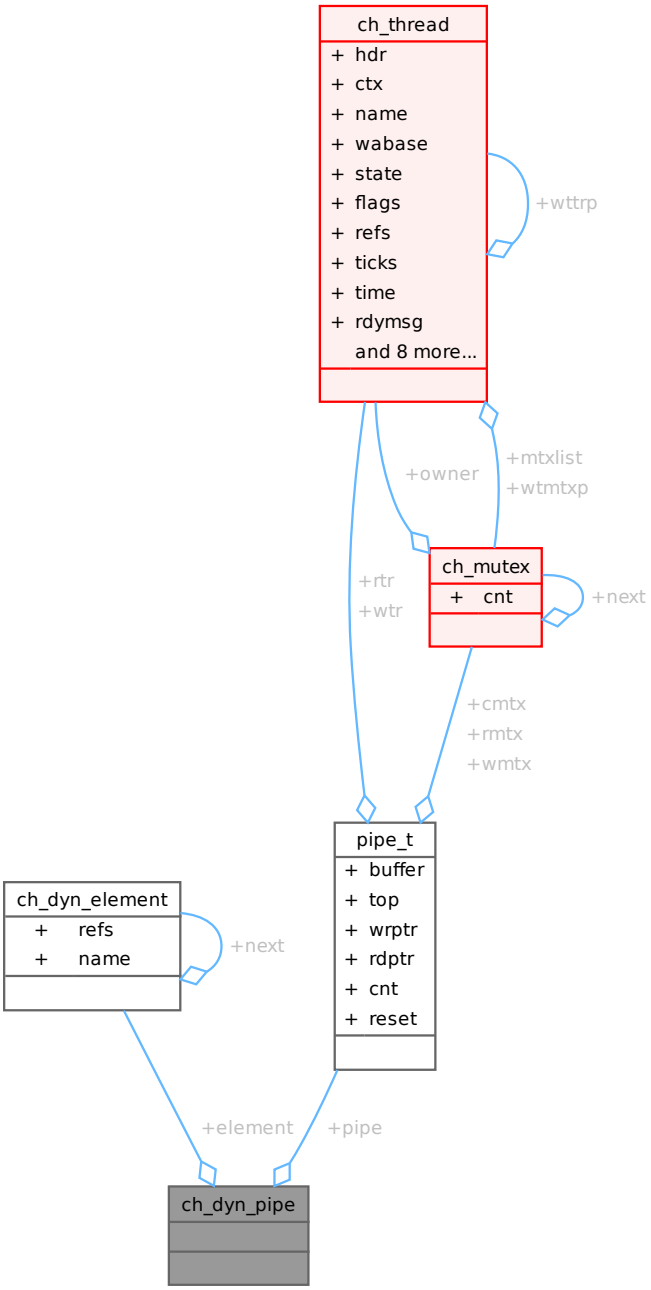
The objects FIFO.

7.8 ch_dyn_pipe Struct Reference

Type of a dynamic pipe object.

```
#include <chfactory.h>
```

Collaboration diagram for ch_dyn_pipe:



Data Fields

- dyn_element_t element

  List element of the dynamic pipe object.
- pipe_t pipe

  The pipe.

7.8.1 Detailed Description

Type of a dynamic pipe object.

7.8.2 Field Documentation

7.8.2.1 element

```
dyn_element_t ch_dyn_pipe::element
```

List element of the dynamic pipe object.

7.8.2.2 pipe

```
pipe_t ch_dyn_pipe::pipe
```

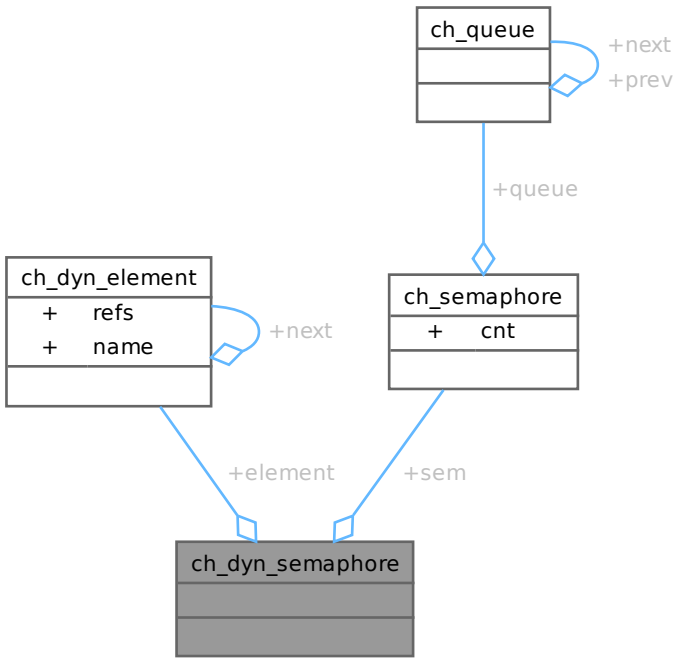
The pipe.

7.9 ch_dyn_semaphore Struct Reference

Type of a dynamic semaphore.

```
#include <chfactory.h>
```

Collaboration diagram for ch_dyn_semaphore:



Data Fields

- dyn_element_t element

  List element of the dynamic semaphore.
- semaphore_t sem

  The semaphore.

7.9.1 Detailed Description

Type of a dynamic semaphore.

7.9.2 Field Documentation

7.9.2.1 element

```
dyn_element_t ch_dyn_semaphore::element
```

List element of the dynamic semaphore.

7.9.2.2 sem

```
semaphore_t ch_dyn_semaphore::sem
```

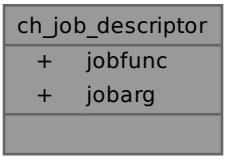
The semaphore.

7.10 ch_job_descriptor Struct Reference

Type of a job descriptor.

```
#include <chjobs.h>
```

Collaboration diagram for ch_job_descriptor:



Data Fields

- job_function_t jobfunc
  Job function.
- void * jobarg
  Argument to be passed to the job function.

7.10.1 Detailed Description

Type of a job descriptor.

7.10.2 Field Documentation

7.10.2.1 jobfunc

```
job_function_t ch_job_descriptor::jobfunc
```

Job function.

7.10.2.2 jobarg

```
void* ch_job_descriptor::jobarg
```

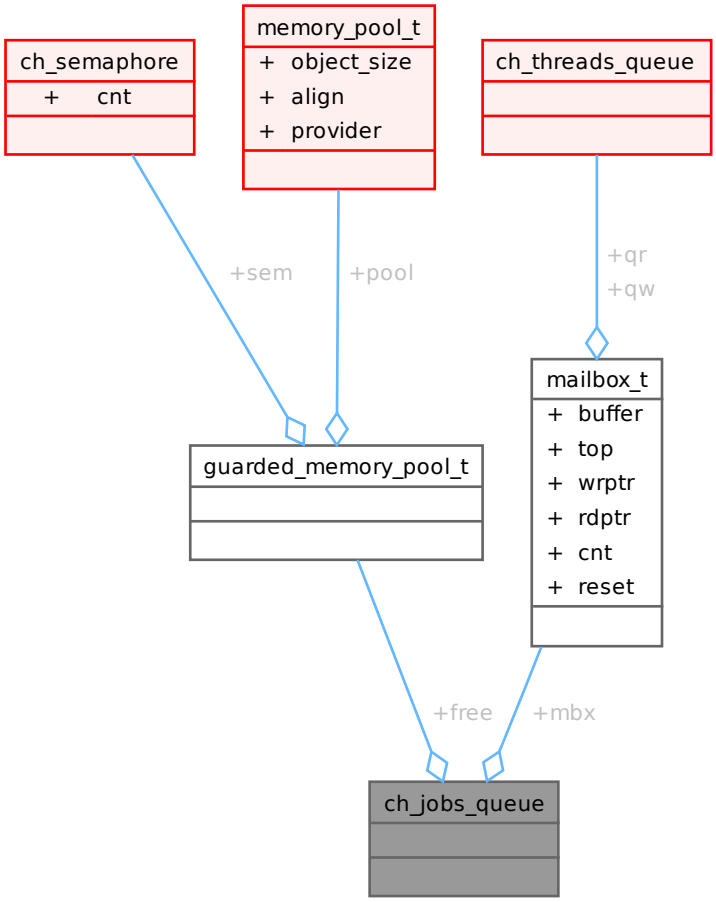
Argument to be passed to the job function.

7.11 ch_jobs_queue Struct Reference

Type of a jobs queue.

```
#include <chjobs.h>
```

Collaboration diagram for ch_jobs_queue:



Data Fields

- guarded_memory_pool_t free
  Pool of the free jobs.
- mailbox_t mbx
  Mailbox of the sent jobs.

7.11.1 Detailed Description

Type of a jobs queue.

7.11.2 Field Documentation

7.11.2.1 free

```
guarded_memory_pool_t ch_jobs_queue::free
```

Pool of the free jobs.

7.11.2.2 mbx

```
mailbox_t ch_jobs_queue::mbx
```

Mailbox of the sent jobs.

7.12 ch_list Struct Reference

Structure representing a generic single link list header and element.

```
#include <chlists.h>
```

Collaboration diagram for ch_list:

Data Fields

- ch_list_t * next
  Next in the list/queue.

7.12.1 Detailed Description

Structure representing a generic single link list header and element.

7.12.2 Field Documentation

7.12.2.1 next

```
ch_list_t* ch_list::next
```

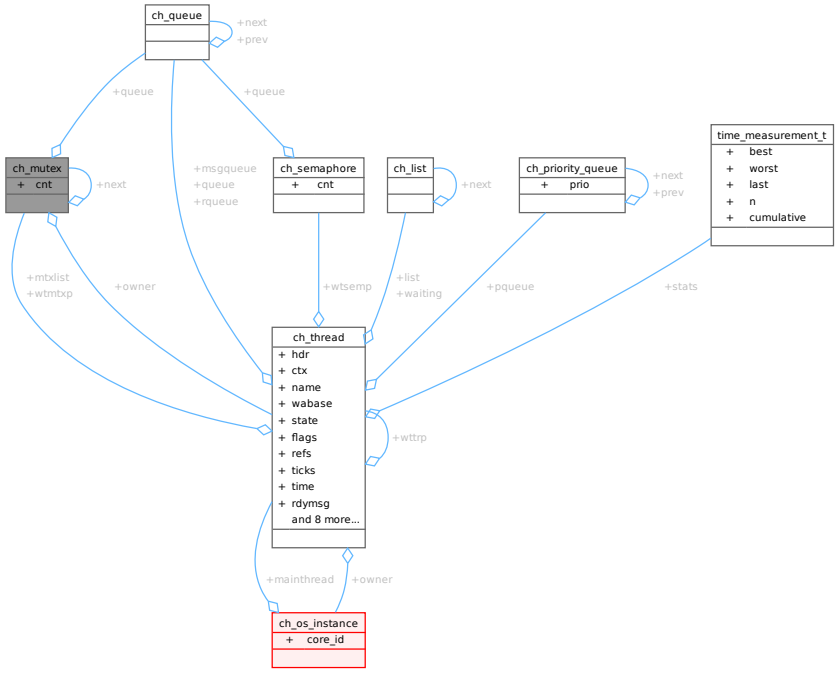
Next in the list/queue.

7.13 ch_mutex Struct Reference

Mutex structure.

`#include <chmtx.h>`

Collaboration diagram for ch_mutex:



Data Fields

- ch_queue_t queue

  Queue of the threads sleeping on this mutex.
- thread_t * owner

  Owner `thread_t` pointer or `NULL`.
- mutex_t * next

  Next `mutex_t` into an owner-list or `NULL`.
- cnt_t cnt

  Mutex recursion counter.

7.13.1 Detailed Description

Mutex structure.

7.13.2 Field Documentation

7.13.2.1 queue

```
ch_queue_t ch_mutex::queue
```

Queue of the threads sleeping on this mutex.

7.13.2.2 owner

```
thread_t* ch_mutex::owner
```

Owner `thread_t` pointer or `NULL`.

7.13.2.3 next

```
mutex_t* ch_mutex::next
```

Next `mutex_t` into an owner-list or `NULL`.

7.13.2.4 cnt

```
cnt_t ch_mutex::cnt
```

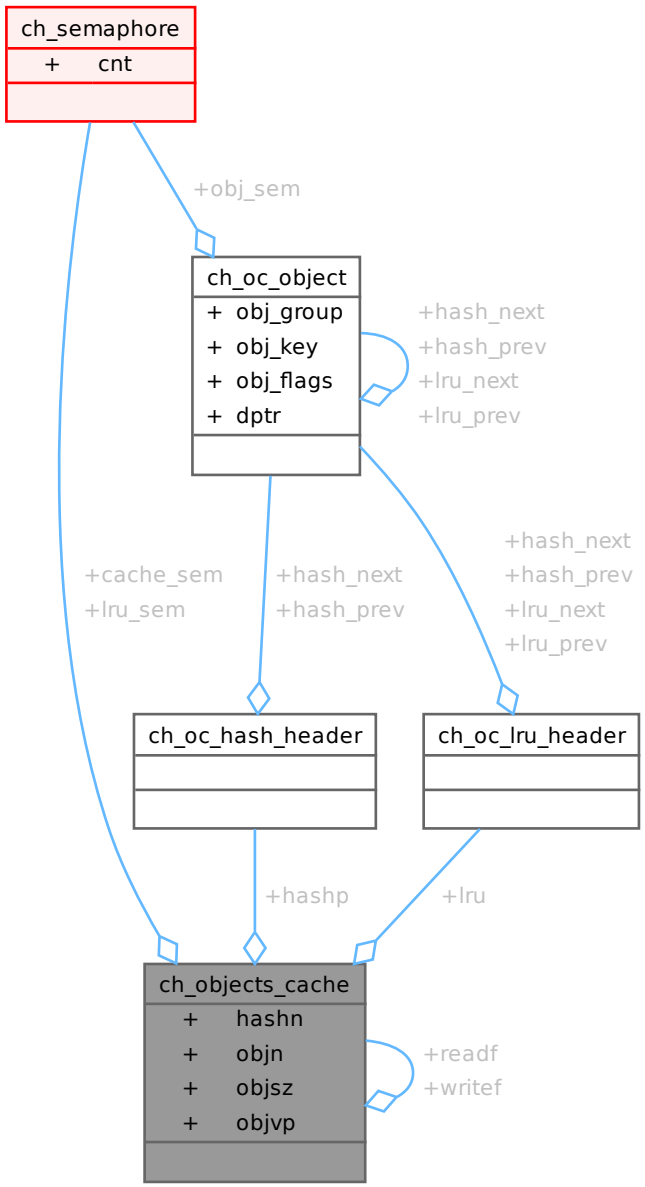
Mutex recursion counter.

7.14 ch_objects_cache Struct Reference

Structure representing a cache object.

```
#include <chobjcaches.h>
```

Collaboration diagram for ch_objects_cache:



Data Fields

- ucnt_t hashn

  Number of elements in the hash table.
- oc_hash_header_t * hashp

  Pointer to the hash table.
- ucnt_t objn

  Number of elements in the objects table.
- size_t objsz

Size of elements in the objects table.

- void * objvp

  Pointer to the objects table.
- oc_lru_header_t lru

  LRU list header.
- semaphore_t cache_sem

  Semaphore for cache access.
- semaphore_t lru_sem

  Semaphore for LRU access.
- oc_readf_t readf

  Reader functions for cached objects.
- oc_writef_t writef

  Writer functions for cached objects.

7.14.1 Detailed Description

Structure representing a cache object.

7.14.2 Field Documentation

7.14.2.1 hashn

```
ucnt_t ch_objects_cache::hashn
```

Number of elements in the hash table.

7.14.2.2 hashp

```
oc_hash_header_t* ch_objects_cache::hashp
```

Pointer to the hash table.

7.14.2.3 objn

```
ucnt_t ch_objects_cache::objn
```

Number of elements in the objects table.

7.14.2.4 objsz

```
size_t ch_objects_cache::objsz
```

Size of elements in the objects table.

7.14.2.5 objvp

```
void* ch_objects_cache::objvp
```

Pointer to the objects table.

7.14.2.6 lru

```
oc_lru_header_t ch_objects_cache::lru
```

LRU list header.

7.14.2.7 cache_sem

```
semaphore_t ch_objects_cache::cache_sem
```

Semaphore for cache access.

7.14.2.8 lru_sem

```
semaphore_t ch_objects_cache::lru_sem
```

Semaphore for LRU access.

7.14.2.9 readf

```
oc_readf_t ch_objects_cache::readf
```

Reader functions for cached objects.

7.14.2.10 writef

```
oc_writef_t ch_objects_cache::writef
```

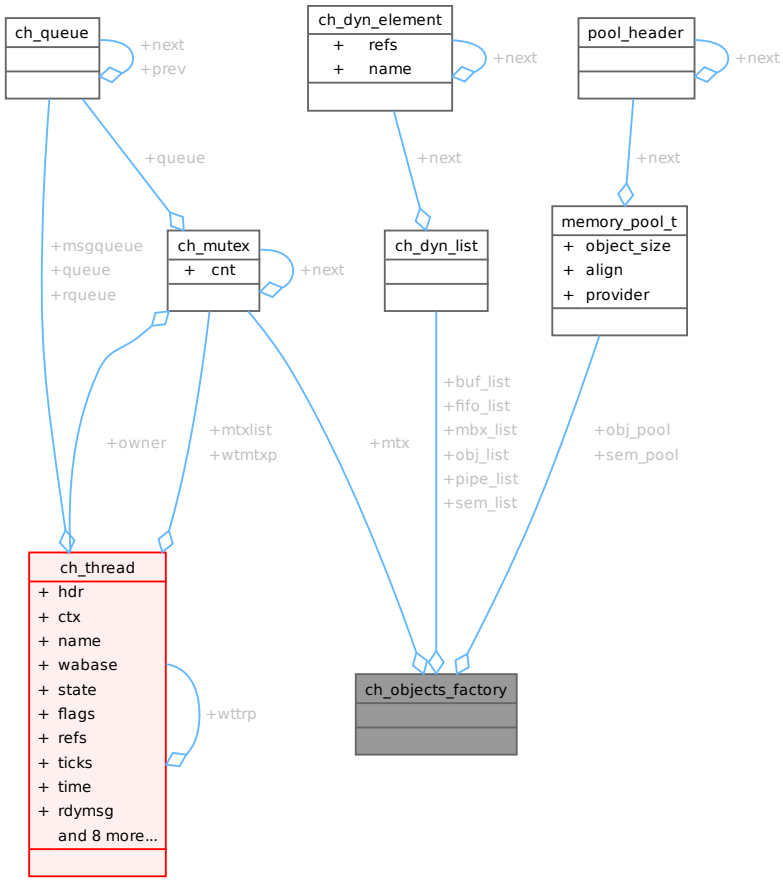
Writer functions for cached objects.

7.15 ch_objects_factory Struct Reference

Type of the factory main object.

`#include <chfactory.h>`

Collaboration diagram for ch_objects_factory:



Data Fields

- mutex_t mtx

  Factory access mutex or semaphore.
- dyn_list_t obj_list

  List of the registered objects.
- memory_pool_t obj_pool

  Pool of the available registered objects.
- dyn_list_t buf_list

  List of the allocated buffer objects.
- dyn_list_t sem_list

  List of the allocated semaphores.
- memory_pool_t sem_pool

Pool of the available semaphores.

- dyn_list_t mbx_list

  List of the allocated buffer objects.

- dyn_list_t fifo_list

  List of the allocated "objects FIFO" objects.

- dyn_list_t pipe_list

  List of the allocated pipe objects.

7.15.1 Detailed Description

Type of the factory main object.

7.15.2 Field Documentation

7.15.2.1 mtx

```
mutex_t ch_objects_factory::mtx
```

Factory access mutex or semaphore.

7.15.2.2 obj_list

```
dyn_list_t ch_objects_factory::obj_list
```

List of the registered objects.

7.15.2.3 obj_pool

```
memory_pool_t ch_objects_factory::obj_pool
```

Pool of the available registered objects.

7.15.2.4 buf_list

```
dyn_list_t ch_objects_factory::buf_list
```

List of the allocated buffer objects.

7.15.2.5 sem_list

```
dyn_list_t ch_objects_factory::sem_list
```

List of the allocated semaphores.

7.15.2.6 sem_pool

```
memory_pool_t ch_objects_factory::sem_pool
```

Pool of the available semaphores.

7.15.2.7 mbx_list

```
dyn_list_t ch_objects_factory::mbx_list
```

List of the allocated buffer objects.

7.15.2.8 fifo_list

```
dyn_list_t ch_objects_factory::fifo_list
```

List of the allocated "objects FIFO" objects.

7.15.2.9 pipe_list

```
dyn_list_t ch_objects_factory::pipe_list
```

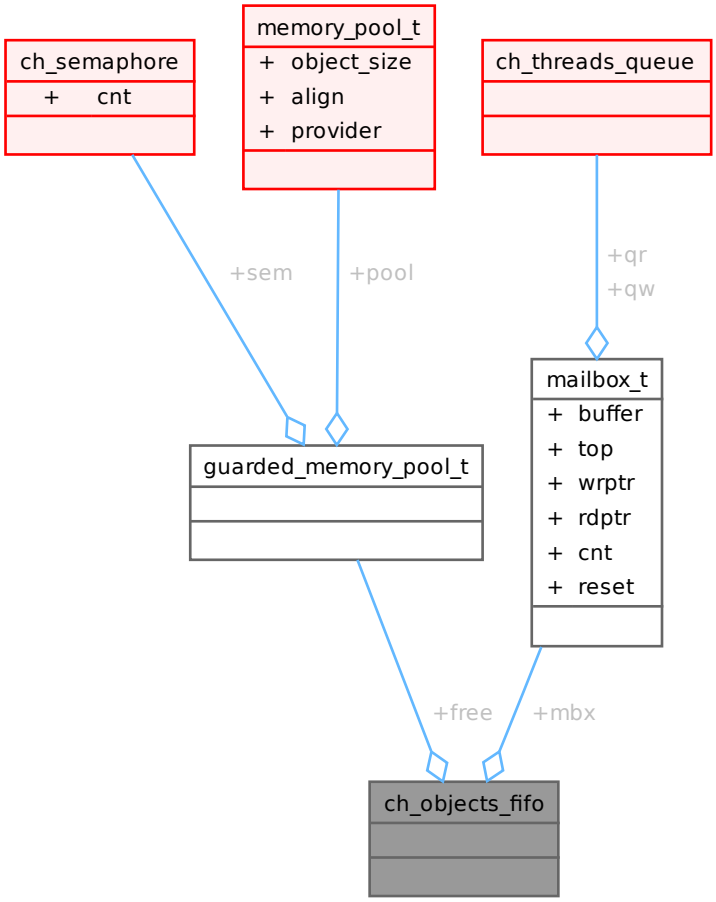
List of the allocated pipe objects.

7.16 ch_objects_fifo Struct Reference

Type of an objects FIFO.

```
#include <chobjfifos.h>
```

Collaboration diagram for ch_objects_fifo:



Data Fields

- guarded_memory_pool_t free
  Pool of the free objects.
- mailbox_t mbx
  Mailbox of the sent objects.

7.16.1 Detailed Description

Type of an objects FIFO.

7.16.2 Field Documentation

7.16.2.1 free

```
guarded_memory_pool_t ch_objects_fifo::free
```

Pool of the free objects.

7.16.2.2 mbx

```
mailbox_t ch_objects_fifo::mbx
```

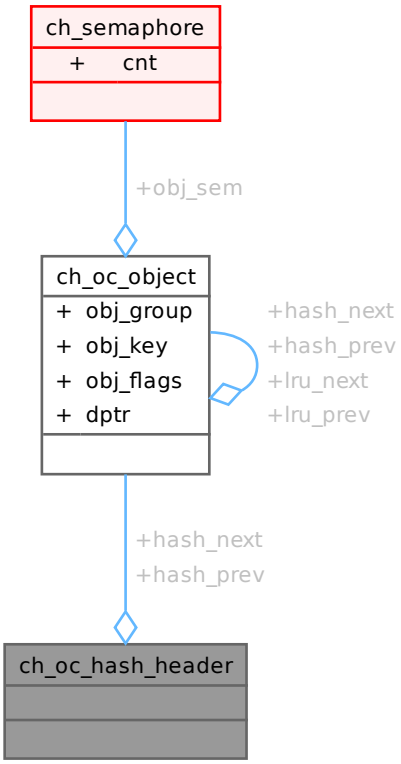
Mailbox of the sent objects.

7.17 ch_oc_hash_header Struct Reference

Structure representing an hash table element.

```
#include <chobjcaches.h>
```

Collaboration diagram for ch_oc_hash_header:



Data Fields

- oc_object_t * hash_next
  Next in the collisions list.
- oc_object_t * hash_prev
  Previous in the collisions list.

7.17.1 Detailed Description

Structure representing an hash table element.

7.17.2 Field Documentation

7.17.2.1 hash_next

```
oc_object_t* ch_oc_hash_header::hash_next
```

Next in the collisions list.

7.17.2.2 hash_prev

```
oc_object_t* ch_oc_hash_header::hash_prev
```

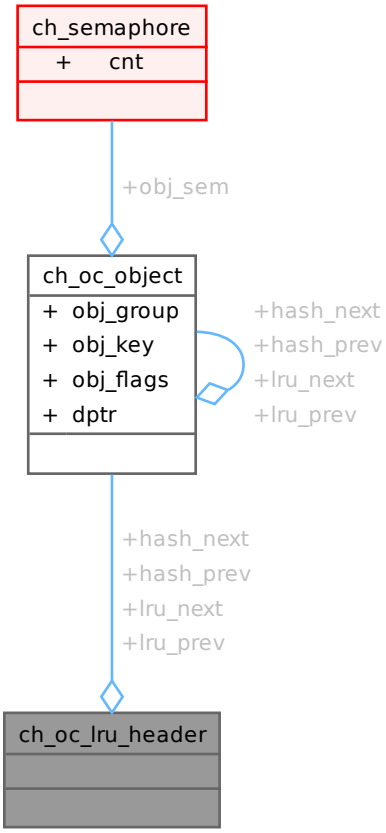
Previous in the collisions list.

7.18 ch_oc_lru_header Struct Reference

Structure representing an hash table element.

```
#include <chobjcaches.h>
```

Collaboration diagram for ch_oc_lru_header:



Data Fields

- oc_object_t * hash_next
  Next in the collisions list.
- oc_object_t * hash_prev
  Previous in the collisions list.
- oc_object_t * lru_next
  Next in the LRU list.
- oc_object_t * lru_prev
  Previous in the LRU list.

7.18.1 Detailed Description

Structure representing an hash table element.

7.18.2 Field Documentation

7.18.2.1 hash_next

```
oc_object_t* ch_oc_lru_header::hash_next
```

Next in the collisions list.

7.18.2.2 hash_prev

```
oc_object_t* ch_oc_lru_header::hash_prev
```

Previous in the collisions list.

7.18.2.3 lru_next

```
oc_object_t* ch_oc_lru_header::lru_next
```

Next in the LRU list.

7.18.2.4 lru_prev

```
oc_object_t* ch_oc_lru_header::lru_prev
```

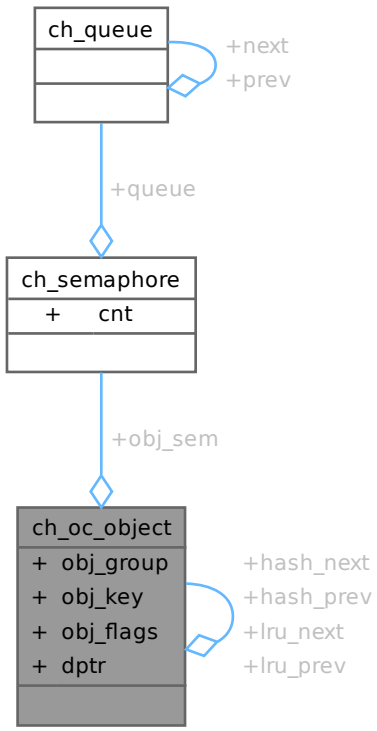
Previous in the LRU list.

7.19 ch_oc_object Struct Reference

Structure representing a cached object.

```
#include <chobjcaches.h>
```

Collaboration diagram for ch_oc_object:



Data Fields

- oc_object_t * hash_next
  Next in the collisions list.
- oc_object_t * hash_prev
  Previous in the collisions list.
- oc_object_t * lru_next
  Next in the LRU list.
- oc_object_t * lru_prev
  Previous in the LRU list.
- uint32_t obj_group
  Object group.
- uint32_t obj_key
  Object key.
- semaphore_t obj_sem
  Semaphore for object access.
- oc_flags_t obj_flags
  Object flags.
- void * dptr
  User pointer.

7.19.1 Detailed Description

Structure representing a cached object.

7.19.2 Field Documentation

7.19.2.1 hash_next

```
oc_object_t* ch_oc_object::hash_next
```

Next in the collisions list.

7.19.2.2 hash_prev

```
oc_object_t* ch_oc_object::hash_prev
```

Previous in the collisions list.

7.19.2.3 lru_next

```
oc_object_t* ch_oc_object::lru_next
```

Next in the LRU list.

7.19.2.4 lru_prev

```
oc_object_t* ch_oc_object::lru_prev
```

Previous in the LRU list.

7.19.2.5 obj_group

```
uint32_t ch_oc_object::obj_group
```

Object group.

7.19.2.6 obj_key

```
uint32_t ch_oc_object::obj_key
```

Object key.

7.19.2.7 obj_sem

```
semaphore_t ch_oc_object::obj_sem
```

Semaphore for object access.

7.19.2.8 obj_flags

```
oc_flags_t ch_oc_object::obj_flags
```

Object flags.

7.19.2.9 dptr

```
void* ch_oc_object::dptr
```

User pointer.

Note

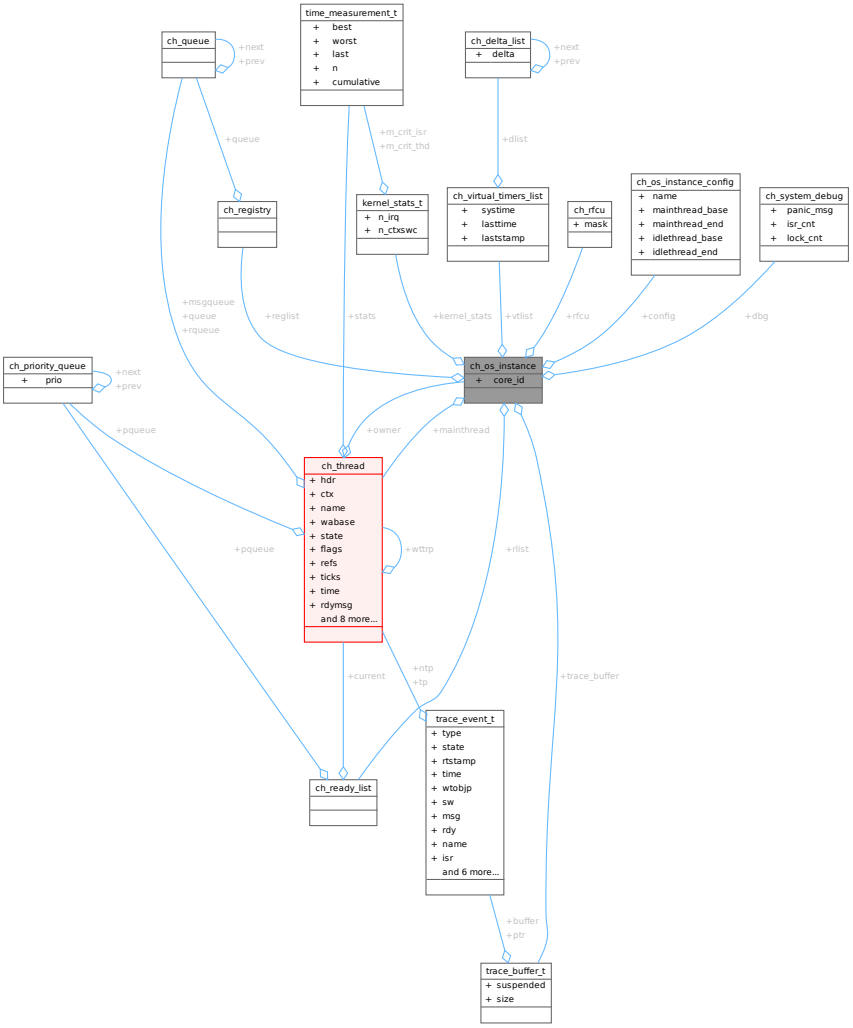
This pointer can be used to refer to external buffers, `chCacheObjectInit()` initializes it to `NULL`.

7.20 ch_os_instance Struct Reference

System instance data structure.

```
#include <chobjects.h>
```

Collaboration diagram for ch_os_instance:



Data Fields

- ready_list_t rlist

  Ready list header.
- virtual_timers_list_t vtlist

  Virtual timers delta list header.
- registry_t reglist

  Registry header.
- core_id_t core_id

  Core associated to this instance.
- rfcu_t rfcu

  Runtime Faults Collection Unit for this instance.
- const os_instance_config_t * config

  Pointer to the instance configuration data.
- thread_t mainthread

  Main thread descriptor.

- system_debug_t dbg
  System debug.
- trace_buffer_t trace_buffer
  Trace buffer.
- kernel_stats_t kernel_stats
  Global kernel statistics.

7.20.1 Detailed Description

System instance data structure.

7.20.2 Field Documentation

7.20.2.1 rlist

```
ready_list_t ch_os_instance::rlist
```

Ready list header.

7.20.2.2 vtlist

```
virtual_timers_list_t ch_os_instance::vtlist
```

Virtual timers delta list header.

7.20.2.3 reglist

```
registry_t ch_os_instance::reglist
```

Registry header.

Note

This field is present only if the SMP mode is disabled.

7.20.2.4 core_id

```
core_id_t ch_os_instance::core_id
```

Core associated to this instance.

7.20.2.5 rfcu

```
rfcu_t ch_os_instance::rfcu
```

Runtime Faults Collection Unit for this instance.

Note

This field is present only if the SMP mode is disabled.

7.20.2.6 config

```
const os_instance_config_t* ch_os_instance::config
```

Pointer to the instance configuration data.

7.20.2.7 mainthread

```
thread_t ch_os_instance::mainthread
```

Main thread descriptor.

7.20.2.8 dbg

```
system_debug_t ch_os_instance::dbg
```

System debug.

7.20.2.9 trace_buffer

```
trace_buffer_t ch_os_instance::trace_buffer
```

Trace buffer.

7.20.2.10 kernel_stats

```
kernel_stats_t ch_os_instance::kernel_stats
```

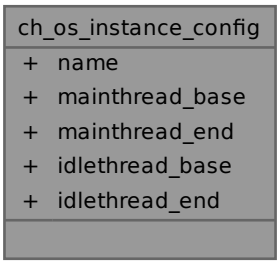
Global kernel statistics.

7.21 ch_os_instance_config Struct Reference

Type of an system instance configuration.

`#include <chobjects.h>`

Collaboration diagram for ch_os_instance_config:



Data Fields

- const char * name
  Instance name.
- stkalign_t * mainthread_base
  Lower limit of the main function thread stack.
- stkalign_t * mainthread_end
  Upper limit of the main function thread stack.
- stkalign_t * idlethread_base
  Lower limit of the dedicated idle thread stack.
- stkalign_t * idlethread_end
  Upper limit of the dedicated idle thread stack.

7.21.1 Detailed Description

Type of an system instance configuration.

7.21.2 Field Documentation

7.21.2.1 name

`const char* ch_os_instance_config::name`

Instance name.

7.21.2.2 mainthread_base

```
stkalign_t* ch_os_instance_config::mainthread_base
```

Lower limit of the main function thread stack.

7.21.2.3 mainthread_end

```
stkalign_t* ch_os_instance_config::mainthread_end
```

Upper limit of the main function thread stack.

7.21.2.4 idlethread_base

```
stkalign_t* ch_os_instance_config::idlethread_base
```

Lower limit of the dedicated idle thread stack.

7.21.2.5 idlethread_end

```
stkalign_t* ch_os_instance_config::idlethread_end
```

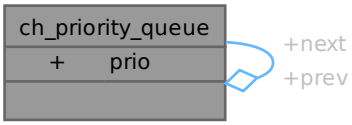
Upper limit of the dedicated idle thread stack.

7.22 ch_priority_queue Struct Reference

Structure representing a generic priority-ordered bidirectional linked list header and element.

```
#include <chlists.h>
```

Collaboration diagram for ch_priority_queue:



Data Fields

- ch_priority_queue_t * next
  Next in the queue.
- ch_priority_queue_t * prev
  Previous in the queue.
- tprio_t prio
  Priority of this element.

7.22.1 Detailed Description

Structure representing a generic priority-ordered bidirectional linked list header and element.

Note

Link fields are void pointers in order to avoid aliasing issues.

7.22.2 Field Documentation

7.22.2.1 next

```
ch_priority_queue_t* ch_priority_queue::next
```

Next in the queue.

7.22.2.2 prev

```
ch_priority_queue_t* ch_priority_queue::prev
```

Previous in the queue.

7.22.2.3 prio

```
tprio_t ch_priority_queue::prio
```

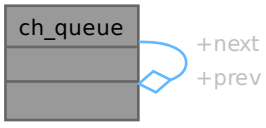
Priority of this element.

7.23 ch_queue Struct Reference

Structure representing a generic bidirectional linked list header and element.

`#include <chlists.h>`

Collaboration diagram for ch_queue:



Data Fields

- ch_queue_t * next
  Next in the list/queue.
- ch_queue_t * prev
  Previous in the queue.

7.23.1 Detailed Description

Structure representing a generic bidirectional linked list header and element.

7.23.2 Field Documentation

7.23.2.1 next

```
ch_queue_t* ch_queue::next
```

Next in the list/queue.

7.23.2.2 prev

```
ch_queue_t* ch_queue::prev
```

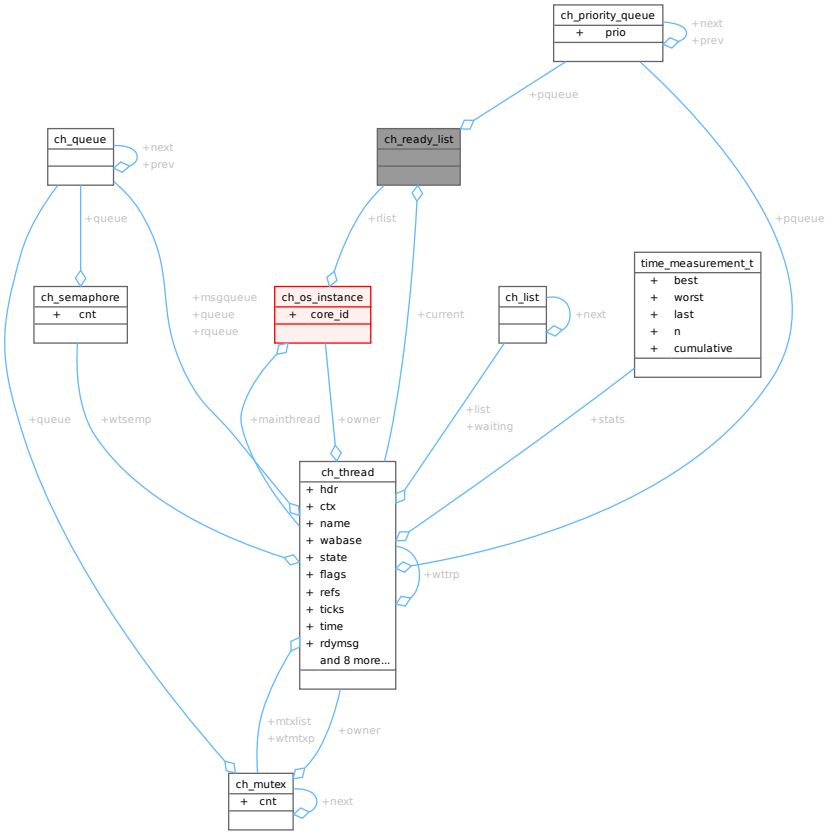
Previous in the queue.

7.24 ch_ready_list Struct Reference

Type of a ready list header.

```
#include <chobjects.h>
```

Collaboration diagram for ch_ready_list:



Data Fields

- ch_priority_queue_t pqueue

  Threads ordered queues header.
- thread_t * current

  The currently running thread.

7.24.1 Detailed Description

Type of a ready list header.

7.24.2 Field Documentation

7.24.2.1 pqueue

```
ch_priority_queue_t ch_ready_list::pqueue
```

Threads ordered queues header.

Note

The priority field must be initialized to zero.

7.24.2.2 current

```
thread_t* ch_ready_list::current
```

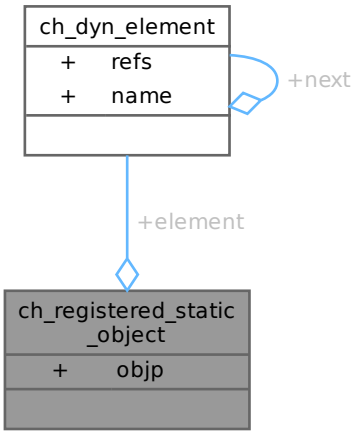
The currently running thread.

7.25 ch_registered_static_object Struct Reference

Type of a registered object.

```
#include <chfactory.h>
```

Collaboration diagram for ch_registered_static_object:



Data Fields

- dyn_element_t element

  List element of the registered object.
- void * objp

  Pointer to the object.

7.25.1 Detailed Description

Type of a registered object.

7.25.2 Field Documentation

7.25.2.1 element

```
dyn_element_t ch_registered_static_object::element
```

List element of the registered object.

7.25.2.2 objp

```
void* ch_registered_static_object::objp
```

Pointer to the object.

Note

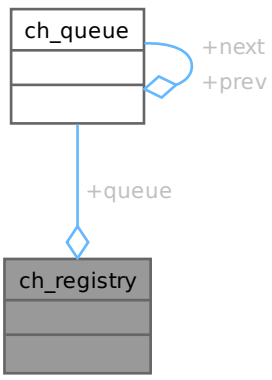
The type of the object is not stored in anyway.

7.26 ch_registry Struct Reference

Type of a registry structure.

```
#include <chobjects.h>
```

Collaboration diagram for ch_registry:



Data Fields

- ch_queue_t queue

  Registry queue header.

7.26.1 Detailed Description

Type of a registry structure.

7.26.2 Field Documentation

7.26.2.1 queue

```
ch_queue_t ch_registry::queue
```

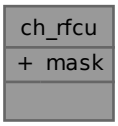
Registry queue header.

7.27 ch_rfcu Struct Reference

Type of an RFCU structure.

```
#include <chrfcu.h>
```

Collaboration diagram for ch_rfcu:



Data Fields

- rfcu_mask_t mask
  Mask of the pending runtime faults.

7.27.1 Detailed Description

Type of an RFCU structure.

7.27.2 Field Documentation

7.27.2.1 mask

```
rfcu_mask_t ch_rfcu::mask
```

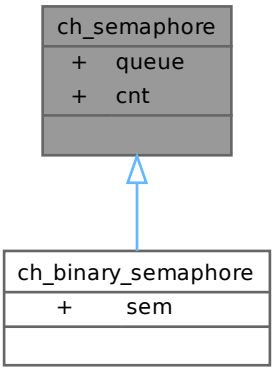
Mask of the pending runtime faults.

7.28 ch_semaphore Struct Reference

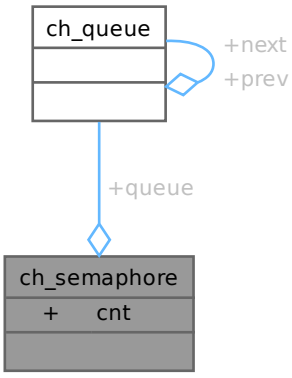
Semaphore structure.

`#include <chsem.h>`

Inheritance diagram for ch_semaphore:



Collaboration diagram for ch_semaphore:



Data Fields

- ch_queue_t queue

  Queue of the threads sleeping on this semaphore.
- cnt_t cnt

  The semaphore counter.

7.28.1 Detailed Description

Semaphore structure.

7.28.2 Field Documentation

7.28.2.1 queue

```
ch_queue_t ch_semaphore::queue
```

Queue of the threads sleeping on this semaphore.

7.28.2.2 cnt

```
cnt_t ch_semaphore::cnt
```

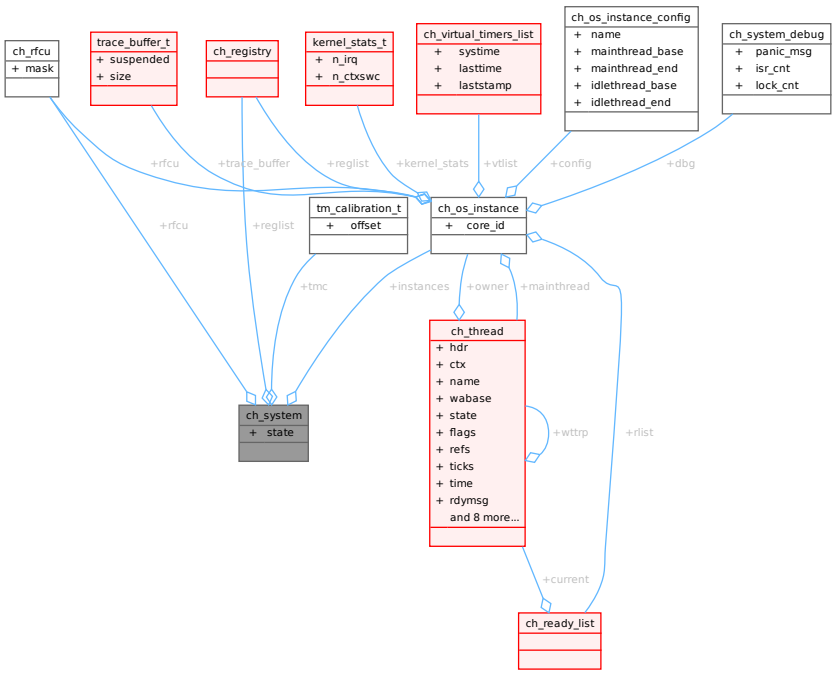
The semaphore counter.

7.29 ch_system Struct Reference

Type of system data structure.

```
#include <chobjects.h>
```

Collaboration diagram for ch_system:



Data Fields

- system_state_t state
  Operating system state.
- os_instance_t * instances [PORT_CORES_NUMBER]
  Initialized OS instances or NULL.
- tm_calibration_t tmc
  Time measurement calibration data.
- registry_t reglist
  Registry header.
- rfcu_t rfcu
  Runtime Faults Collection Unit.

7.29.1 Detailed Description

Type of system data structure.

7.29.2 Field Documentation

7.29.2.1 state

```
system_state_t ch_system::state
```

Operating system state.

7.29.2.2 instances

```
os_instance_t* ch_system::instances[PORT_CORES_NUMBER]
```

Initialized OS instances or NULL.

7.29.2.3 tmc

```
tm_calibration_t ch_system::tmc
```

Time measurement calibration data.

7.29.2.4 reglist

```
registry_t ch_system::reglist
```

Registry header.

Note

This field is present only if the SMP mode is enabled.

7.29.2.5 rfcu

```
rfcu_t ch_system::rfcu
```

Runtime Faults Collection Unit.

Note

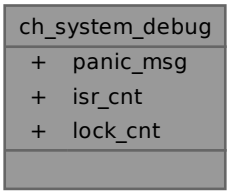
This field is present only if the SMP mode is enabled.

7.30 ch_system_debug Struct Reference

System debug data structure.

```
#include <chdebug.h>
```

Collaboration diagram for ch_system_debug:



Data Fields

- const char *volatile panic_msg

  Pointer to the panic message.
- cnt_t isr_cnt

  ISR nesting level.
- cnt_t lock_cnt

  Lock nesting level.

7.30.1 Detailed Description

System debug data structure.

7.30.2 Field Documentation

7.30.2.1 panic_msg

```
const char* volatile ch_system_debug::panic_msg
```

Pointer to the panic message.

This pointer is meant to be accessed through the debugger, it is written once and then the system is halted.

Note

Accesses to this pointer must never be optimized out so the field itself is declared volatile.

7.30.2.2 isr_cnt

```
cnt_t ch_system_debug::isr_cnt
```

ISR nesting level.

7.30.2.3 lock_cnt

```
cnt_t ch_system_debug::lock_cnt
```

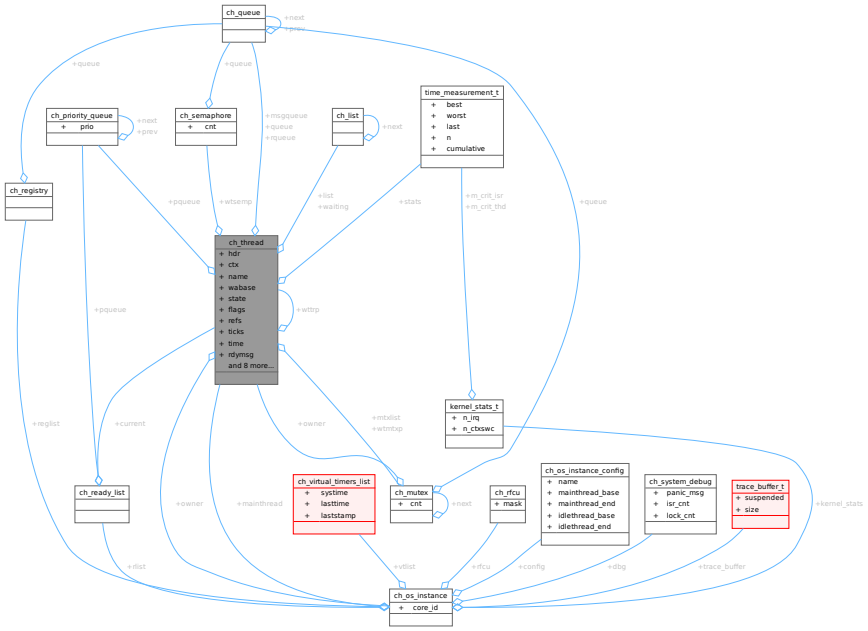
Lock nesting level.

7.31 ch_thread Struct Reference

Structure representing a thread.

```
#include <chobjects.h>
```

Collaboration diagram for ch_thread:



Data Fields

- union {
  ch_list_t list
  Threads lists element.
  ch_queue_t queue
  Threads queues element.
  ch_priority_queue_t pqueue
  Threads ordered queues element.
  } hdr

  Shared list headers.
- struct port_context ctx

  Processor context.
- ch_queue_t rqueue

  Registry queue element.
- os_instance_t * owner

  OS instance owner of this thread.
- const char * name

  Thread name or `NULL`.
- stkalign_t * wabase

  Working area base address.
- tstate_t state

  Current thread state.
- tmode_t flags

  Various thread flags.
- trefs_t refs

  References to this thread.
- tslices_t ticks

  Number of ticks remaining to this thread.
- volatile systime_t time

  Thread consumed time in ticks.
- union {
  msg_t rdymsg
  Thread wakeup code.
  msg_t exitcode
  Thread exit code.
  void * wtobjp
  Pointer to a generic "wait" object.
  thread_reference_t * wttrp
  Pointer to a generic thread reference object.
  msg_t sentmsg
  Thread sent message.
  struct ch_semaphore * wtsemp
  Pointer to a generic semaphore object.
  struct ch_mutex * wtmtxp
  Pointer to a generic mutex object.
  eventmask_t ewmask
  Enabled events mask.
  } u

  State-specific fields.
- ch_list_t waiting

  Termination waiting list.

- ch_queue_t msgqueue

  Messages queue.
- eventmask_t epending

  Pending events mask.
- struct ch_mutex * mtxlist

  List of the mutexes owned by this thread.
- tprio_t realprio

  Thread's own, non-inherited, priority.
- void * mpool

  Memory Pool where the thread workspace is returned.
- time_measurement_t stats

  Thread statistics.

7.31.1 Detailed Description

Structure representing a thread.

Note

Not all the listed fields are always needed, by switching off some not needed ChibiOS/RT subsystems it is possible to save RAM space by shrinking this structure.

7.31.2 Field Documentation

7.31.2.1 list

```
ch_list_t ch_thread::list
```

Threads lists element.

7.31.2.2 queue

```
ch_queue_t ch_thread::queue
```

Threads queues element.

7.31.2.3 pqueue

```
ch_priority_queue_t ch_thread::pqueue
```

Threads ordered queues element.

7.31.2.4 [union]

```
union { ... } ch_thread::hdr
```

Shared list headers.

7.31.2.5 ctx

```
struct port_context ch_thread::ctx
```

Processor context.

7.31.2.6 rqueue

```
ch_queue_t ch_thread::rqueue
```

Registry queue element.

7.31.2.7 owner

```
os_instance_t* ch_thread::owner
```

OS instance owner of this thread.

7.31.2.8 name

```
const char* ch_thread::name
```

Thread name or NULL.

7.31.2.9 wabase

```
stkalign_t* ch_thread::wabase
```

Working area base address.

Note

This pointer is used for stack overflow checks and for dynamic threading.

7.31.2.10 state

```
tstate_t ch_thread::state
```

Current thread state.

7.31.2.11 flags

```
tmode_t ch_thread::flags
```

Various thread flags.

7.31.2.12 refs

```
trefs_t ch_thread::refs
```

References to this thread.

7.31.2.13 ticks

```
tslices_t ch_thread::ticks
```

Number of ticks remaining to this thread.

7.31.2.14 time

```
volatile systime_t ch_thread::time
```

Thread consumed time in ticks.

Note

This field can overflow.

7.31.2.15 rdymsg

```
msg_t ch_thread::rdymsg
```

Thread wakeup code.

Note

This field contains the low level message sent to the thread by the waking thread or interrupt handler. The value is valid after exiting the `chSchWakeupS()` function.

7.31.2.16 exitcode

```
msg_t ch_thread::exitcode
```

Thread exit code.

Note

The thread termination code is stored in this field in order to be retrieved by the thread performing a `chThdWait()` on this thread.

7.31.2.17 wtobjp

```
void* ch_thread::wtobjp
```

Pointer to a generic "wait" object.

Note

This field is used to get a generic pointer to a synchronization object and is valid when the thread is in one of the wait states.

7.31.2.18 wttrp

```
thread_reference_t* ch_thread::wttrp
```

Pointer to a generic thread reference object.

Note

This field is used to get a pointer to a synchronization object and is valid when the thread is in CH_STATE↩_SUSPENDED state.

7.31.2.19 sentmsg

```
msg_t ch_thread::sentmsg
```

Thread sent message.

7.31.2.20 wtsemp

```
struct ch_semaphore* ch_thread::wtsemp
```

Pointer to a generic semaphore object.

Note

This field is used to get a pointer to a synchronization object and is valid when the thread is in CH_STATE↩_WTSEM state.

7.31.2.21 wtmtxp

```
struct ch_mutex* ch_thread::wtmtxp
```

Pointer to a generic mutex object.

Note

This field is used to get a pointer to a synchronization object and is valid when the thread is in CH_STATE↩_WTMTX state.

7.31.2.22 ewmask

```
eventmask_t ch_thread::ewmask
```

Enabled events mask.

Note

This field is only valid while the thread is in the CH_STATE_WTOREVT or CH_STATE_WTANDEVT states.

7.31.2.23 [union]

```
union { ... } ch_thread::u
```

State-specific fields.

Note

All the fields declared in this union are only valid in the specified state or condition and are thus volatile.

7.31.2.24 waiting

```
ch_list_t ch_thread::waiting
```

Termination waiting list.

7.31.2.25 msgqueue

```
ch_queue_t ch_thread::msgqueue
```

Messages queue.

7.31.2.26 epending

```
eventmask_t ch_thread::epending
```

Pending events mask.

7.31.2.27 mtxlist

```
struct ch_mutex* ch_thread::mtxlist
```

List of the mutexes owned by this thread.

Note

The list is terminated by a NULL in this field.

7.31.2.28 realprio

```
tprio_t ch_thread::realprio
```

Thread's own, non-inherited, priority.

7.31.2.29 mpool

```
void* ch_thread::mpool
```

Memory Pool where the thread workspace is returned.

7.31.2.30 stats

```
time_measurement_t ch_thread::stats
```

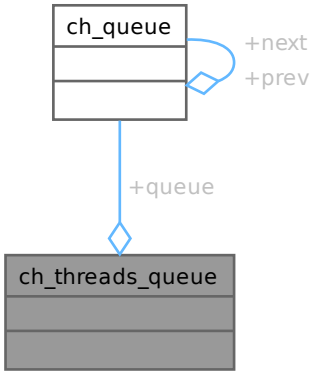
Thread statistics.

7.32 ch_threads_queue Struct Reference

Type of a threads queue.

```
#include <chobjects.h>
```

Collaboration diagram for ch_threads_queue:



Data Fields

- ch_queue_t queue

  Threads queue header.

7.32.1 Detailed Description

Type of a threads queue.

7.32.2 Field Documentation

7.32.2.1 queue

```
ch_queue_t ch_threads_queue::queue
```

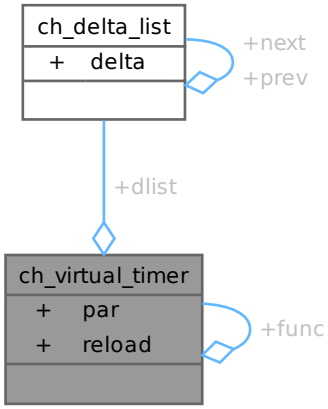
Threads queue header.

7.33 ch_virtual_timer Struct Reference

Structure representing a Virtual Timer.

```
#include <chobjects.h>
```

Collaboration diagram for ch_virtual_timer:



Data Fields

- ch_delta_list_t dlist
  Delta list element.
- vtfunc_t func
  Timer callback function pointer.
- void * par
  Timer callback function parameter.
- sysinterval_t reload
  Current reload interval.

7.33.1 Detailed Description

Structure representing a Virtual Timer.

7.33.2 Field Documentation

7.33.2.1 dlist

```
ch_delta_list_t ch_virtual_timer::dlist
```

Delta list element.

7.33.2.2 func

```
vtfunc_t ch_virtual_timer::func
```

Timer callback function pointer.

7.33.2.3 par

```
void* ch_virtual_timer::par
```

Timer callback function parameter.

7.33.2.4 reload

```
sysinterval_t ch_virtual_timer::reload
```

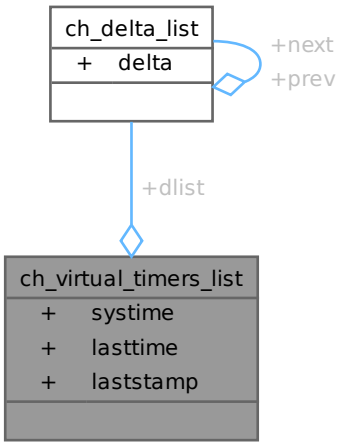
Current reload interval.

7.34 ch_virtual_timers_list Struct Reference

Type of virtual timers list header.

```
#include <chobjects.h>
```

Collaboration diagram for ch_virtual_timers_list:



Data Fields

- ch_delta_list_t dlist
  Delta list header.
- volatile systime_t systime
  System Time counter.
- systime_t lasttime
  System time of the last tick event.
- volatile uint64_t laststamp
  Last generated time stamp.

7.34.1 Detailed Description

Type of virtual timers list header.

Note

The timers list is implemented as a double link bidirectional list in order to make the unlink time constant, the reset of a virtual timer is often used in the code.

7.34.2 Field Documentation

7.34.2.1 dlist

```
ch_delta_list_t ch_virtual_timers_list::dlist
```

Delta list header.

7.34.2.2 systime

```
volatile systime_t ch_virtual_timers_list::systime
```

System Time counter.

7.34.2.3 lasttime

```
systime_t ch_virtual_timers_list::lasttime
```

System time of the last tick event.

7.34.2.4 laststamp

```
volatile uint64_t ch_virtual_timers_list::laststamp
```

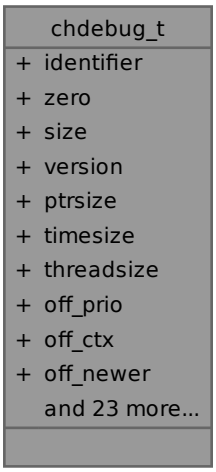
Last generated time stamp.

7.35 chdebug_t Struct Reference

ChibiOS/RT memory signature record.

```
#include <chregistry.h>
```

Collaboration diagram for chdebug_t:



Data Fields

- char identifier [4]
  Always set to "main".
- uint8_t zero
  Must be zero.
- uint8_t size
  Size of this structure.
- uint16_t version
  Encoded ChibiOS/RT version.
- uint8_t ptrsize
  Size of a pointer.
- uint8_t timesize
  Size of a `systime_t`.
- uint8_t threadsize
  Size of a `thread_t`.
- uint8_t off_prio
  Offset of `prio` *field.*
- uint8_t off_ctx
  Offset of `ctx` *field.*
- uint8_t off_newer
  Offset of `newer` *field.*
- uint8_t off_older
  Offset of `older` *field.*
- uint8_t off_name
  Offset of `name` *field.*
- uint8_t off_stklimit
  Offset of `stklimit` *field.*
- uint8_t off_state
  Offset of `state` *field.*
- uint8_t off_flags
  Offset of `flags` *field.*
- uint8_t off_refs
  Offset of `refs` *field.*
- uint8_t off_preempt
  Offset of `ticks` *field.*
- uint8_t off_time
  Offset of `time` *field.*
- uint8_t off_reserved [4]
- uint8_t intctxsize
  Size of a `port_intctx`.
- uint8_t intervalsize
  Size of a `sysinterval_t`.
- uint8_t instancesnum
  Number of instances.
- uint8_t off_sys_state
  Offset of `state` *field.*
- uint8_t off_sys_instances
  Offset of `instances` *array field.*
- uint8_t off_sys_reglist
  Offset of `reglist` *field.*

- uint8_t off_sys_rfcu
  Offset of `rfcu` field.
- uint8_t off_sys_reserved [4]
- uint8_t off_inst_rlist_current
  Offset of `rlist.current` field.
- uint8_t off_inst_rlist
  Offset of `rlist` field.
- uint8_t off_inst_vtlist
  Offset of `vtlist` field.
- uint8_t off_inst_reglist
  Offset of `reglist` field.
- uint8_t off_inst_core_id
  Offset of `core_id` field.
- uint8_t off_inst_rfcu
  Offset of `rfcu` field.

7.35.1 Detailed Description

ChibiOS/RT memory signature record.

7.35.2 Field Documentation

7.35.2.1 identifier

```
char chdebug_t::identifier[4]
```

Always set to "main".

7.35.2.2 zero

```
uint8_t chdebug_t::zero
```

Must be zero.

7.35.2.3 size

```
uint8_t chdebug_t::size
```

Size of this structure.

7.35.2.4 version

```
uint16_t chdebug_t::version
```

Encoded ChibiOS/RT version.

7.35.2.5 ptrsize

```
uint8_t chdebug_t::ptrsize
```

Size of a pointer.

7.35.2.6 timesize

```
uint8_t chdebug_t::timesize
```

Size of a systime_t.

7.35.2.7 threadsize

```
uint8_t chdebug_t::threadsize
```

Size of a thread_t.

7.35.2.8 off_prio

```
uint8_t chdebug_t::off_prio
```

Offset of prio field.

7.35.2.9 off_ctx

```
uint8_t chdebug_t::off_ctx
```

Offset of ctx field.

7.35.2.10 off_newer

```
uint8_t chdebug_t::off_newer
```

Offset of newer field.

7.35.2.11 off_older

```
uint8_t chdebug_t::off_older
```

Offset of older field.

7.35.2.12 off_name

```
uint8_t chdebug_t::off_name
```

Offset of name field.

7.35.2.13 off_stklimit

```
uint8_t chdebug_t::off_stklimit
```

Offset of `stklimit` field.

7.35.2.14 off_state

```
uint8_t chdebug_t::off_state
```

Offset of `state` field.

7.35.2.15 off_flags

```
uint8_t chdebug_t::off_flags
```

Offset of `flags` field.

7.35.2.16 off_refs

```
uint8_t chdebug_t::off_refs
```

Offset of `refs` field.

7.35.2.17 off_preempt

```
uint8_t chdebug_t::off_preempt
```

Offset of `ticks` field.

7.35.2.18 off_time

```
uint8_t chdebug_t::off_time
```

Offset of `time` field.

7.35.2.19 off_reserved

```
uint8_t chdebug_t::off_reserved[4]
```

7.35.2.20 intctxsize

```
uint8_t chdebug_t::intctxsize
```

Size of a `port_intctx`.

7.35.2.21 intervalsize

```
uint8_t chdebug_t::intervalsize
```

Size of a sysinterval_t.

7.35.2.22 instancesnum

```
uint8_t chdebug_t::instancesnum
```

Number of instances.

7.35.2.23 off_sys_state

```
uint8_t chdebug_t::off_sys_state
```

Offset of state field.

7.35.2.24 off_sys_instances

```
uint8_t chdebug_t::off_sys_instances
```

Offset of instances array field.

7.35.2.25 off_sys_reglist

```
uint8_t chdebug_t::off_sys_reglist
```

Offset of reglist field.

7.35.2.26 off_sys_rfcu

```
uint8_t chdebug_t::off_sys_rfcu
```

Offset of rfcu field.

7.35.2.27 off_sys_reserved

```
uint8_t chdebug_t::off_sys_reserved[4]
```

7.35.2.28 off_inst_rlist_current

```
uint8_t chdebug_t::off_inst_rlist_current
```

Offset of rlist.current field.

7.35.2.29 off_inst_rlist

`uint8_t chdebug_t::off_inst_rlist`

Offset of `rlist` field.

7.35.2.30 off_inst_vtlist

`uint8_t chdebug_t::off_inst_vtlist`

Offset of `vtlist` field.

7.35.2.31 off_inst_reglist

`uint8_t chdebug_t::off_inst_reglist`

Offset of `reglist` field.

7.35.2.32 off_inst_core_id

`uint8_t chdebug_t::off_inst_core_id`

Offset of `core_id` field.

7.35.2.33 off_inst_rfcu

`uint8_t chdebug_t::off_inst_rfcu`

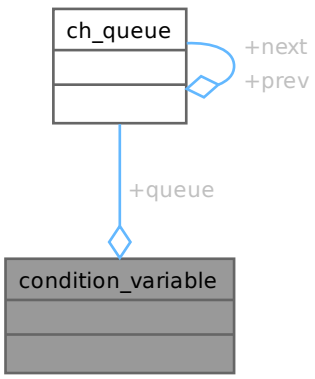
Offset of `rfcu` field.

7.36 condition_variable Struct Reference

condition_variable_t structure.

`#include <chcond.h>`

Collaboration diagram for condition_variable:



Data Fields

- ch_queue_t queue
  Condition variable threads queue.

7.36.1 Detailed Description

condition_variable_t structure.

7.36.2 Field Documentation

7.36.2.1 queue

```
ch_queue_t condition_variable::queue
```

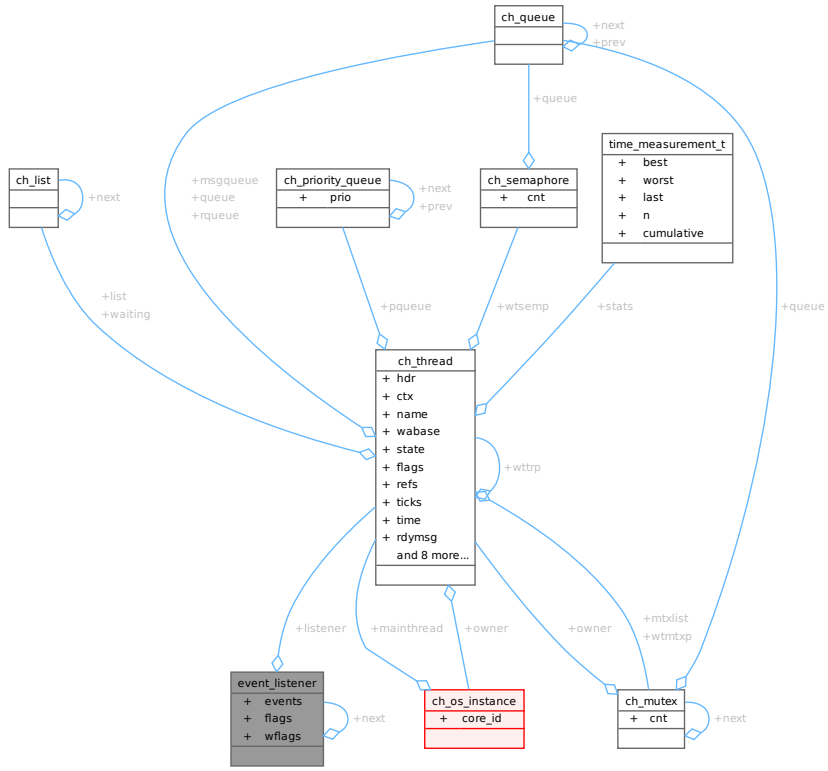
Condition variable threads queue.

7.37 event_listener Struct Reference

Event Listener structure.

```
#include <chevents.h>
```

Collaboration diagram for event_listener:



Data Fields

- event_listener_t * next
  Next Event Listener registered on the event source.
- thread_t * listener
  Thread interested in the event source.
- eventmask_t events
  Events to be set in the listening thread.
- eventflags_t flags
  Flags added to the listener by the event source.
- eventflags_t wflags
  Flags that this listener interested in.

7.37.1 Detailed Description

Event Listener structure.

7.37.2 Field Documentation

7.37.2.1 next

```
event_listener_t* event_listener::next
```

Next Event Listener registered on the event source.

7.37.2.2 listener

```
thread_t* event_listener::listener
```

Thread interested in the event source.

7.37.2.3 events

```
eventmask_t event_listener::events
```

Events to be set in the listening thread.

7.37.2.4 flags

```
eventflags_t event_listener::flags
```

Flags added to the listener by the event source.

7.37.2.5 wflags

```
eventflags_t event_listener::wflags
```

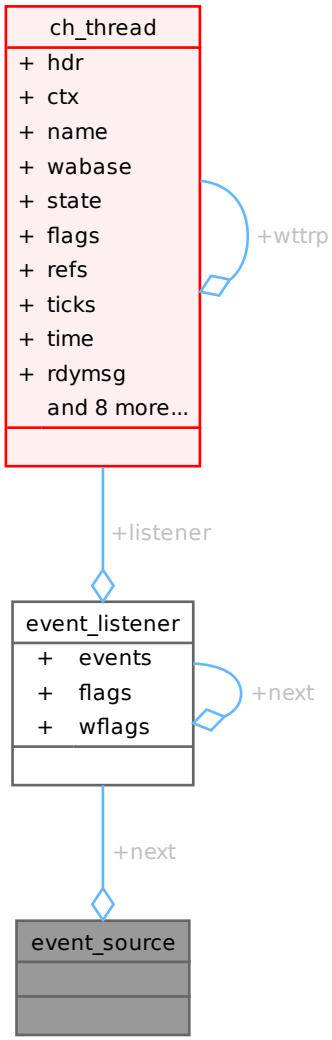
Flags that this listener interested in.

7.38 event_source Struct Reference

Event Source structure.

```
#include <chevents.h>
```

Collaboration diagram for event_source:



Data Fields

- event_listener_t * next

  First Event Listener registered on the Event Source.

7.38.1 Detailed Description

Event Source structure.

7.38.2 Field Documentation

7.38.2.1 next

```
event_listener_t* event_source::next
```

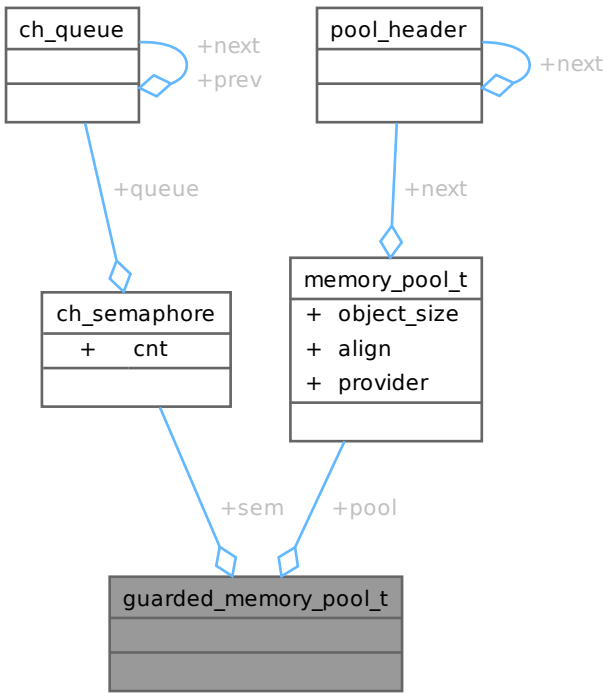
First Event Listener registered on the Event Source.

7.39 guarded_memory_pool_t Struct Reference

Guarded memory pool descriptor.

```
#include <chmempools.h>
```

Collaboration diagram for guarded_memory_pool_t:



Data Fields

- semaphore_t sem

  Counter semaphore guarding the memory pool.
- memory_pool_t pool

  The memory pool itself.

7.39.1 Detailed Description

Guarded memory pool descriptor.

7.39.2 Field Documentation

7.39.2.1 sem

```
semaphore_t guarded_memory_pool_t::sem
```

Counter semaphore guarding the memory pool.

7.39.2.2 pool

```
memory_pool_t guarded_memory_pool_t::pool
```

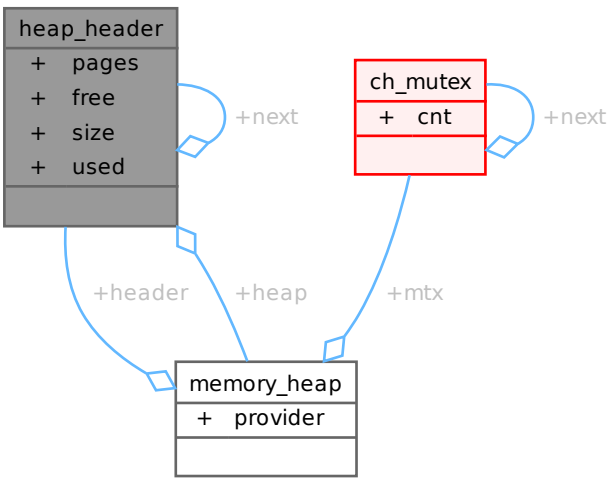
The memory pool itself.

7.40 heap_header Union Reference

Memory heap block header.

`#include <chmemheaps.h>`

Collaboration diagram for heap_header:



Data Fields

- struct {
  heap_header_t * next
  Next block in free list.
  size_t pages
  Size of the area in pages.
  } free

- struct {
  memory_heap_t * heap
  Block owner heap.
  size_t size
  Size of the area in bytes.
  } used

7.40.1 Detailed Description

Memory heap block header.

7.40.2 Field Documentation

7.40.2.1 next

```
heap_header_t* heap_header::next
```

Next block in free list.

7.40.2.2 pages

```
size_t heap_header::pages
```

Size of the area in pages.

7.40.2.3 [struct]

```
struct { ... } heap_header::free
```

7.40.2.4 heap

```
memory_heap_t* heap_header::heap
```

Block owner heap.

7.40.2.5 size

```
size_t heap_header::size
```

Size of the area in bytes.

7.40.2.6 [struct]

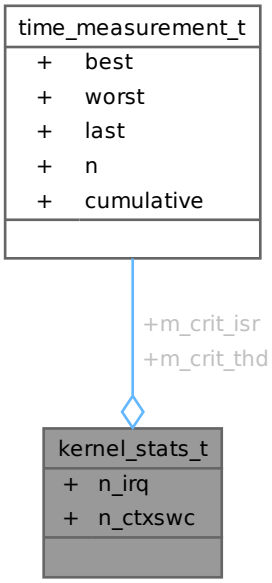
```
struct { ... } heap_header::used
```

7.41 kernel_stats_t Struct Reference

Type of a kernel statistics structure.

```
#include <chstats.h>
```

Collaboration diagram for kernel_stats_t:



Data Fields

- ucnt_t n_irq
  Number of IRQs.
- ucnt_t n_ctxswc
  Number of context switches.
- time_measurement_t m_crit_thd
  Measurement of threads critical zones duration.
- time_measurement_t m_crit_isr
  Measurement of ISRs critical zones duration.

7.41.1 Detailed Description

Type of a kernel statistics structure.

7.41.2 Field Documentation

7.41.2.1 n_irq

```
ucnt_t kernel_stats_t::n_irq
```

Number of IRQs.

7.41.2.2 n_ctxswc

```
ucnt_t kernel_stats_t::n_ctxswc
```

Number of context switches.

7.41.2.3 m_crit_thd

```
time_measurement_t kernel_stats_t::m_crit_thd
```

Measurement of threads critical zones duration.

7.41.2.4 m_crit_isr

```
time_measurement_t kernel_stats_t::m_crit_isr
```

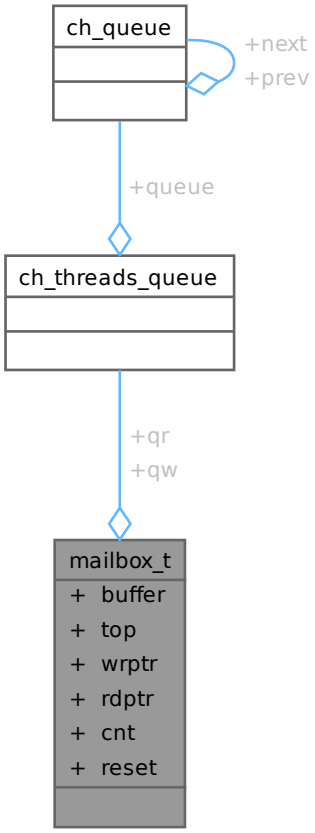
Measurement of ISRs critical zones duration.

7.42 mailbox_t Struct Reference

Structure representing a mailbox object.

```
#include <chmboxes.h>
```

Collaboration diagram for mailbox_t:



Data Fields

- msg_t * buffer
  Pointer to the mailbox buffer.
- msg_t * top
  Pointer to the location after the buffer.
- msg_t * wrptr
  Write pointer.
- msg_t * rdptr
  Read pointer.
- size_t cnt
  Messages in queue.
- bool reset
  True in reset state.
- threads_queue_t qw
  Queued writers.
- threads_queue_t qr
  Queued readers.

7.42.1 Detailed Description

Structure representing a mailbox object.

7.42.2 Field Documentation

7.42.2.1 buffer

```
msg_t* mailbox_t::buffer
```

Pointer to the mailbox buffer.

7.42.2.2 top

```
msg_t* mailbox_t::top
```

Pointer to the location after the buffer.

7.42.2.3 wrptr

```
msg_t* mailbox_t::wrptr
```

Write pointer.

7.42.2.4 rdptr

```
msg_t* mailbox_t::rdptr
```

Read pointer.

7.42.2.5 cnt

```
size_t mailbox_t::cnt
```

Messages in queue.

7.42.2.6 reset

```
bool mailbox_t::reset
```

True in reset state.

7.42.2.7 qw

```
threads_queue_t mailbox_t::qw
```

Queued writers.

7.42.2.8 qr

```
threads_queue_t mailbox_t::qr
```

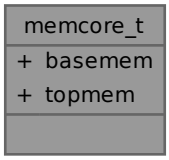
Queued readers.

7.43 memcore_t Struct Reference

Type of memory core object.

```
#include <chmemcore.h>
```

Collaboration diagram for memcore_t:



Data Fields

- uint8_t * basemem
  Next free address.
- uint8_t * topmem
  Final address.

7.43.1 Detailed Description

Type of memory core object.

7.43.2 Field Documentation

7.43.2.1 basemem

```
uint8_t* memcore_t::basemem
```

Next free address.

7.43.2.2 topmem

```
uint8_t* memcore_t::topmem
```

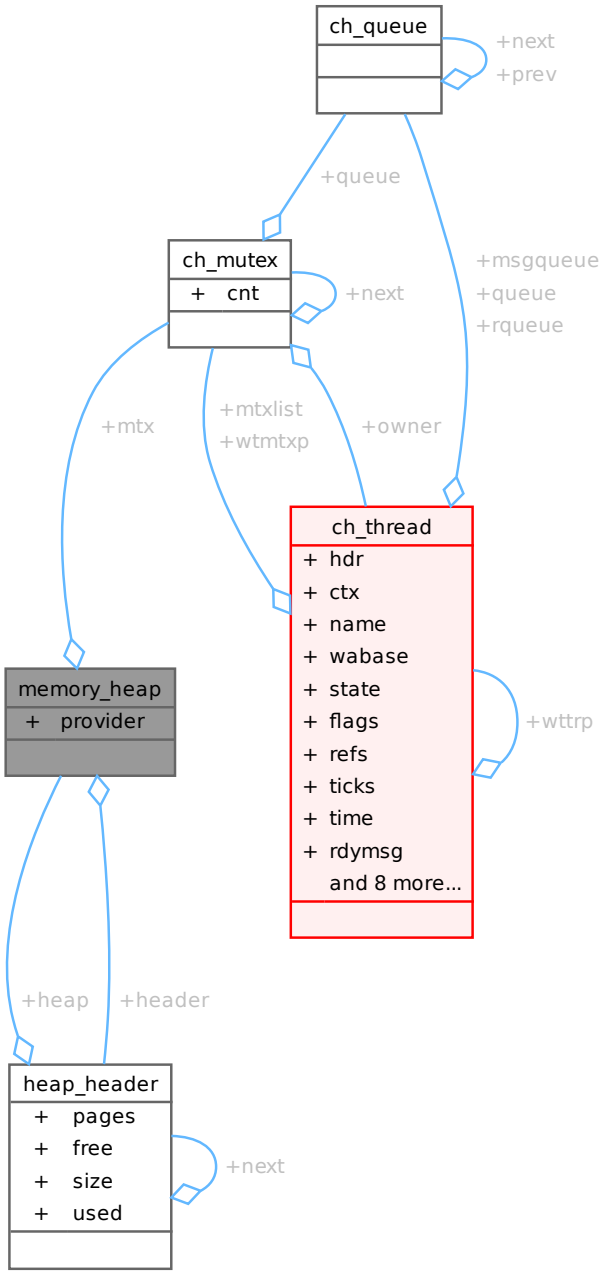
Final address.

7.44 memory_heap Struct Reference

Structure describing a memory heap.

```
#include <chmemheaps.h>
```

Collaboration diagram for memory_heap:



Data Fields

- memgetfunc2_t provider

  Memory blocks provider for this heap.
- heap_header_t header

  Free blocks list header.
- mutex_t mtx

  Heap access mutex.

7.44.1 Detailed Description

Structure describing a memory heap.

7.44.2 Field Documentation

7.44.2.1 provider

```
memgetfunc2_t memory_heap::provider
```

Memory blocks provider for this heap.

7.44.2.2 header

```
heap_header_t memory_heap::header
```

Free blocks list header.

7.44.2.3 mtx

```
mutex_t memory_heap::mtx
```

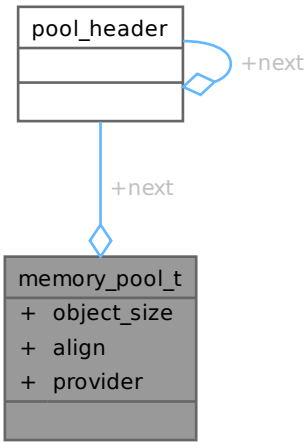
Heap access mutex.

7.45 memory_pool_t Struct Reference

Memory pool descriptor.

```
#include <chmempools.h>
```

Collaboration diagram for memory_pool_t:



Data Fields

- struct pool_header * next
  Pointer to the header.
- size_t object_size
  Memory pool objects size.
- unsigned align
  Required alignment.
- memgetfunc_t provider
  Memory blocks provider for this pool.

7.45.1 Detailed Description

Memory pool descriptor.

7.45.2 Field Documentation

7.45.2.1 next

```
struct pool_header* memory_pool_t::next
```

Pointer to the header.

7.45.2.2 object_size

```
size_t memory_pool_t::object_size
```

Memory pool objects size.

7.45.2.3 align

```
unsigned memory_pool_t::align
```

Required alignment.

7.45.2.4 provider

```
memgetfunc_t memory_pool_t::provider
```

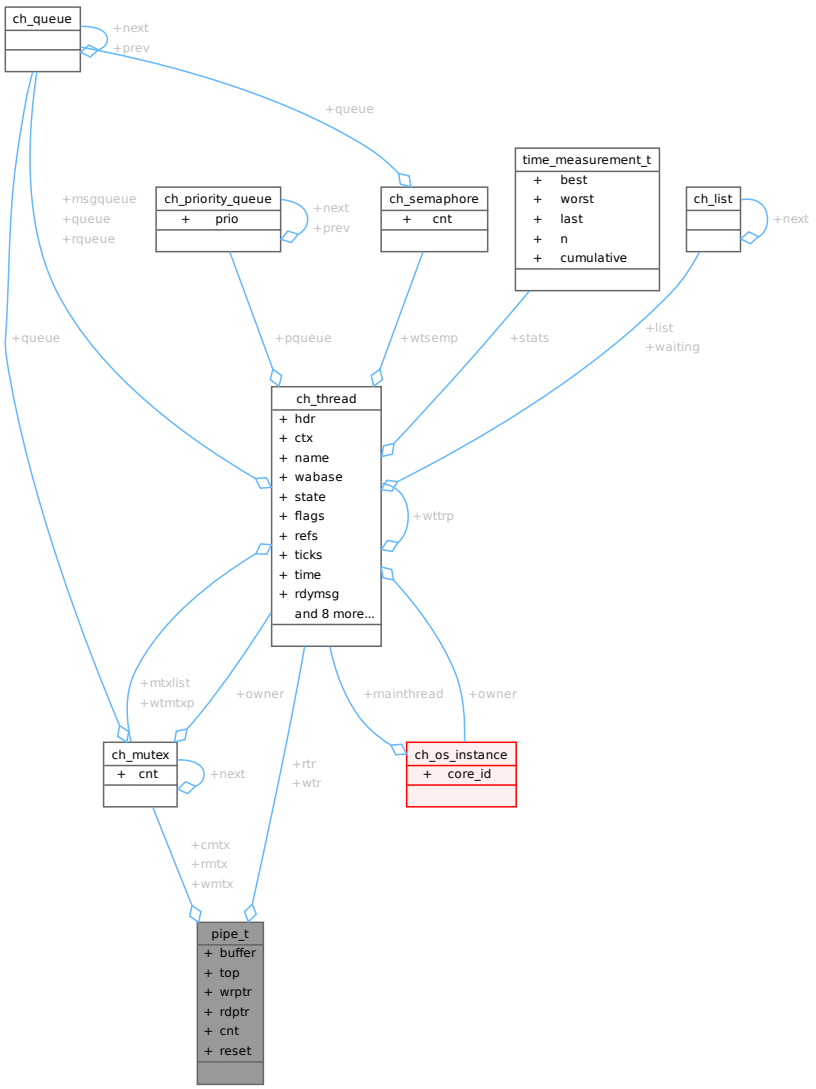
Memory blocks provider for this pool.

7.46 pipe_t Struct Reference

Structure representing a pipe object.

`#include <chpipes.h>`

Collaboration diagram for pipe_t:



Data Fields

- uint8_t * buffer

  Pointer to the pipe buffer.
- uint8_t * top

  Pointer to the location after the buffer.
- uint8_t * wrptr

  Write pointer.

- uint8_t * rdptr
  Read pointer.
- size_t cnt
  Bytes in the pipe.
- bool reset
  True if in reset state.
- thread_reference_t wtr
  Waiting writer.
- thread_reference_t rtr
  Waiting reader.
- mutex_t cmtx
  Common access mutex.
- mutex_t wmtx
  Write access mutex.
- mutex_t rmtx
  Read access mutex.

7.46.1 Detailed Description

Structure representing a pipe object.

7.46.2 Field Documentation

7.46.2.1 buffer

```
uint8_t* pipe_t::buffer
```

Pointer to the pipe buffer.

7.46.2.2 top

```
uint8_t* pipe_t::top
```

Pointer to the location after the buffer.

7.46.2.3 wrptr

```
uint8_t* pipe_t::wrptr
```

Write pointer.

7.46.2.4 rdptr

```
uint8_t* pipe_t::rdptr
```

Read pointer.

7.46.2.5 cnt

```
size_t pipe_t::cnt
```

Bytes in the pipe.

7.46.2.6 reset

```
bool pipe_t::reset
```

True if in reset state.

7.46.2.7 wtr

```
thread_reference_t pipe_t::wtr
```

Waiting writer.

7.46.2.8 rtr

```
thread_reference_t pipe_t::rtr
```

Waiting reader.

7.46.2.9 cmtx

```
mutex_t pipe_t::cmtx
```

Common access mutex.

7.46.2.10 wmtx

```
mutex_t pipe_t::wmtx
```

Write access mutex.

7.46.2.11 rmtx

```
mutex_t pipe_t::rmtx
```

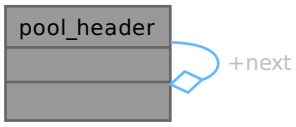
Read access mutex.

7.47 pool_header Struct Reference

Memory pool free object header.

`#include <chmempools.h>`

Collaboration diagram for pool_header:



Data Fields

- struct pool_header * next
  Pointer to the next pool header in the list.

7.47.1 Detailed Description

Memory pool free object header.

7.47.2 Field Documentation

7.47.2.1 next

`struct pool_header* pool_header::next`

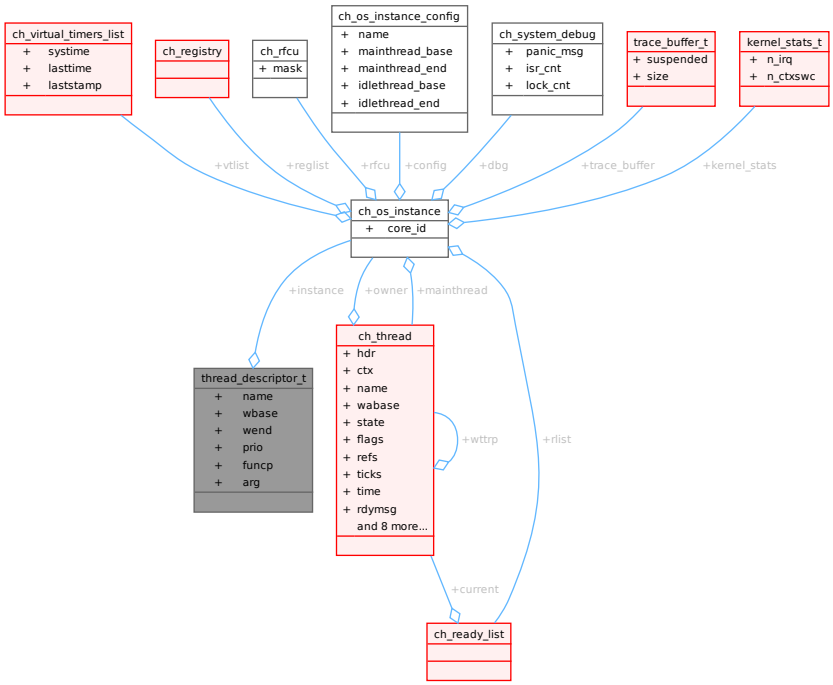
Pointer to the next pool header in the list.

7.48 thread_descriptor_t Struct Reference

Type of a thread descriptor.

`#include <chthreads.h>`

Collaboration diagram for thread_descriptor_t:



Data Fields

- const char * name

  Thread name.
- stkalign_t * wbase

  Pointer to the working area base.
- stkalign_t * wend

  Pointer to the working area end.
- tprio_t prio

  Thread priority.
- tfunc_t funcp

  Thread function pointer.
- void * arg

  Thread argument.
- os_instance_t * instance

  OS instance affinity or `NULL` for current one.

7.48.1 Detailed Description

Type of a thread descriptor.

7.48.2 Field Documentation

7.48.2.1 name

```
const char* thread_descriptor_t::name
```

Thread name.

7.48.2.2 wbase

```
stkalign_t* thread_descriptor_t::wbase
```

Pointer to the working area base.

7.48.2.3 wend

```
stkalign_t* thread_descriptor_t::wend
```

Pointer to the working area end.

7.48.2.4 prio

```
tprio_t thread_descriptor_t::prio
```

Thread priority.

7.48.2.5 funcp

```
tfunc_t thread_descriptor_t::funcp
```

Thread function pointer.

7.48.2.6 arg

```
void* thread_descriptor_t::arg
```

Thread argument.

7.48.2.7 instance

```
os_instance_t* thread_descriptor_t::instance
```

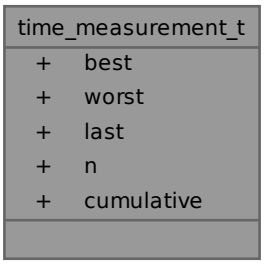
OS instance affinity or `NULL` for current one.

7.49 time_measurement_t Struct Reference

Type of a Time Measurement object.

`#include <chtm.h>`

Collaboration diagram for time_measurement_t:



Data Fields

- rtcnt_t best
  Best measurement.
- rtcnt_t worst
  Worst measurement.
- rtcnt_t last
  Last measurement.
- ucnt_t n
  Number of measurements.
- rttime_t cumulative
  Cumulative measurement.

7.49.1 Detailed Description

Type of a Time Measurement object.

Note

The maximum measurable time period depends on the implementation of the realtime counter and its clock frequency.

The measurement is not 100% cycle-accurate, it can be in excess of few cycles depending on the compiler and target architecture.

Interrupts can affect measurement if the measurement is performed with interrupts enabled.

7.49.2 Field Documentation

7.49.2.1 best

```
rtcnt_t time_measurement_t::best
```

Best measurement.

7.49.2.2 worst

```
rtcnt_t time_measurement_t::worst
```

Worst measurement.

7.49.2.3 last

```
rtcnt_t time_measurement_t::last
```

Last measurement.

7.49.2.4 n

```
ucnt_t time_measurement_t::n
```

Number of measurements.

7.49.2.5 cumulative

```
rttime_t time_measurement_t::cumulative
```

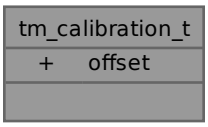
Cumulative measurement.

7.50 tm_calibration_t Struct Reference

Type of a time measurement calibration data.

```
#include <chtm.h>
```

Collaboration diagram for tm_calibration_t:



Data Fields

- rtcnt_t offset

  Measurement calibration value.

7.50.1 Detailed Description

Type of a time measurement calibration data.

7.50.2 Field Documentation

7.50.2.1 offset

```
rtcnt_t tm_calibration_t::offset
```

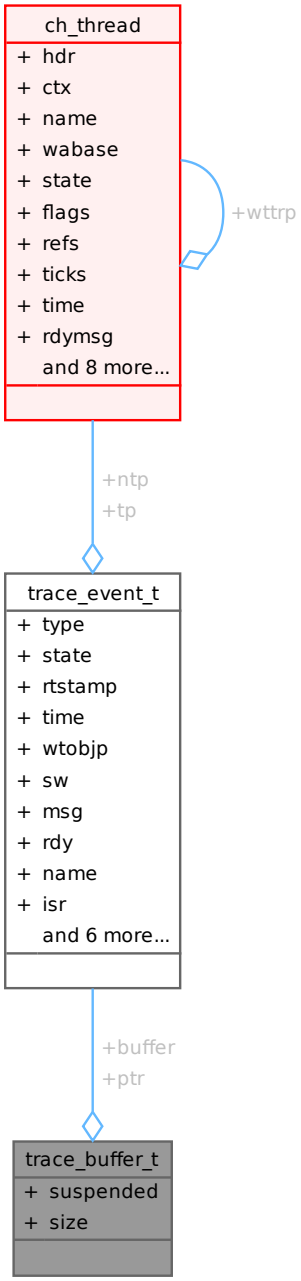
Measurement calibration value.

7.51 trace_buffer_t Struct Reference

Trace buffer header.

```
#include <chtrace.h>
```

Collaboration diagram for trace_buffer_t:



Data Fields

- uint16_t suspended

  Suspended trace sources mask.
- uint16_t size

  Trace buffer size (entries).
- trace_event_t * ptr

Pointer to the buffer front.

- trace_event_t buffer [CH_DBG_TRACE_BUFFER_SIZE]

  Ring buffer.

7.51.1 Detailed Description

Trace buffer header.

7.51.2 Field Documentation

7.51.2.1 suspended

```
uint16_t trace_buffer_t::suspended
```

Suspended trace sources mask.

7.51.2.2 size

```
uint16_t trace_buffer_t::size
```

Trace buffer size (entries).

7.51.2.3 ptr

```
trace_event_t* trace_buffer_t::ptr
```

Pointer to the buffer front.

7.51.2.4 buffer

```
trace_event_t trace_buffer_t::buffer[CH_DBG_TRACE_BUFFER_SIZE]
```

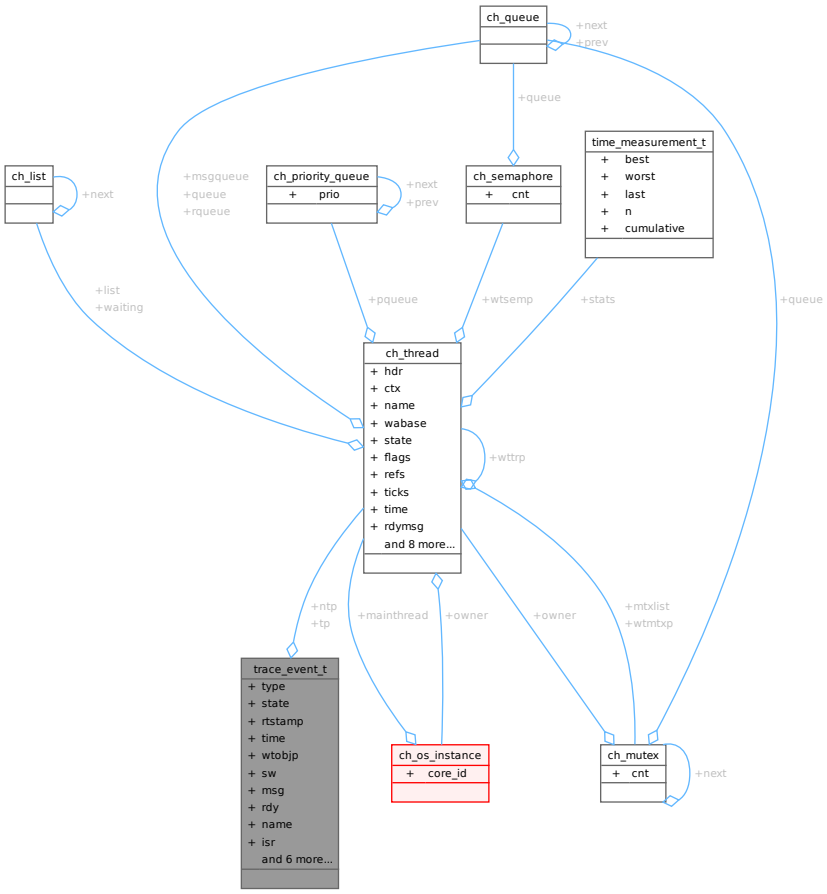
Ring buffer.

7.52 trace_event_t Struct Reference

Trace buffer record.

`#include <chtrace.h>`

Collaboration diagram for trace_event_t:



Data Fields

- uint32_t type:3

  Record type.
- uint32_t state:5

  Switched out thread state.
- uint32_t rtstamp:24

  Accurate time stamp.
- systime_t time

  System time stamp of the switch event.

- union {
  struct {
  thread_t * ntp
  Switched in thread.
  void * wtobjp
  Object where going to sleep.
  } sw
  Structure representing a context switch.
  struct {
  thread_t * tp
  Thread made ready.
  msg_t msg
  Ready message.
  } rdy
  Structure representing a thread becoming ready.
  struct {
  const char * name
  ISR function name taken using `__func__`.
  } isr
  Structure representing an ISR enter.
  struct {
  const char * reason
  Halt error string.
  } halt
  Structure representing an halt.
  struct {
  void * up1
  Trace user parameter 1.
  void * up2
  Trace user parameter 2.
  } user
  User trace structure.
  } u

7.52.1 Detailed Description

Trace buffer record.

7.52.2 Field Documentation

7.52.2.1 type

`uint32_t trace_event_t::type`

Record type.

7.52.2.2 state

`uint32_t trace_event_t::state`

Switched out thread state.

7.52.2.3 rtstamp

```
uint32_t trace_event_t::rtstamp
```

Accurate time stamp.

Note

This field only available if the post supports `PORT_SUPPORTS_RT` else it is set to zero.

7.52.2.4 time

```
systime_t trace_event_t::time
```

System time stamp of the switch event.

7.52.2.5 ntp

```
thread_t* trace_event_t::ntp
```

Switched in thread.

7.52.2.6 wtobjp

```
void* trace_event_t::wtobjp
```

Object where going to sleep.

7.52.2.7 [struct]

```
struct { ... } trace_event_t::sw
```

Structure representing a context switch.

7.52.2.8 tp

```
thread_t* trace_event_t::tp
```

Thread made ready.

7.52.2.9 msg

```
msg_t trace_event_t::msg
```

Ready message.

7.52.2.10 [struct]

```
struct { ... } trace_event_t::rdy
```

Structure representing a thread becoming ready.

7.52.2.11 name

```
const char* trace_event_t::name
```

ISR function name taken using __func__.

7.52.2.12 [struct]

```
struct { ... } trace_event_t::isr
```

Structure representing an ISR enter.

7.52.2.13 reason

```
const char* trace_event_t::reason
```

Halt error string.

7.52.2.14 [struct]

```
struct { ... } trace_event_t::halt
```

Structure representing an halt.

7.52.2.15 up1

```
void* trace_event_t::up1
```

Trace user parameter 1.

7.52.2.16 up2

```
void* trace_event_t::up2
```

Trace user parameter 2.

7.52.2.17 [struct]

```
struct { ... } trace_event_t::user
```

User trace structure.

7.52.2.18 [union]

```
union { ... } trace_event_t::u
```

Chapter 8

File Documentation

8.1 concepts.dox File Reference

8.2 main.dox File Reference

8.3 ccportab.h File Reference

Compiler portability layer.

Macros

Compiler abstraction macros

- #define CC_SECTION(s)
  Allocates a variable or function to a specific section.
- #define CC_WEAK __attribute__((weak))
  Marks a function or variable as a weak symbol.
- #define CC_USED __attribute__((used))
  Marks a function or variable as used.
- #define CC_ALIGN_DATA(n)
  Enforces alignment of the variable declared afterward.
- #define CC_ALIGN_CODE(n)
  Enforces alignment of a function declared afterward.
- #define CC_PACK __attribute__((packed))
  Enforces packing of the structure declared afterward.
- #define CC_NO_INLINE __attribute__((noinline))
  Marks a function as not inlineable.
- #define CC_FORCE_INLINE __attribute__((always_inline))
  Enforces a function inline.
- #define CC_NO_RETURN __attribute__((noreturn))
  Marks a function as non-returning.
- #define CC_ROMCONST const
  Enforces a variable in a ROM area.
- #define CC_LIKELY(x)
  Marks a boolean expression as likely true.
- #define CC_UNLIKELY(x)
  Marks a boolean expression as likely false.

8.3.1 Detailed Description

Compiler portability layer.

8.4 chcustomer.h File Reference

Customer-related info.

Macros

- #define CH_CUSTOMER_ID_STRING "Santa, North Pole"

  Customer readable identifier.
- #define CH_CUSTOMER_ID_CODE "xxxx-yyyy"

  Customer code.
- #define CH_CUSTOMER_LICENSE_EOS_DATE 209912

  End-Of-Support date (yyyymm).
- #define CH_CUSTOMER_LICENSE_VERSION_YEAR 99

  Licensed branch year.
- #define CH_CUSTOMER_LICENSE_VERSION_MONTH 12

  Licensed branch month.
- #define CH_LICENSE CH_LICENSE_GPL

  Current license.
- #define CH_CUSTOMER_LICENSE_VERSION_DATE

  Licensed version date in numeric form (yyyymm).

Licensed Products

- #define CH_CUSTOMER_LIC_RT TRUE
- #define CH_CUSTOMER_LIC_NIL TRUE
- #define CH_CUSTOMER_LIC_OSLIB TRUE
- #define CH_CUSTOMER_LIC_EX TRUE
- #define CH_CUSTOMER_LIC_SB TRUE
- #define CH_CUSTOMER_LIC_PORT_CM0 TRUE
- #define CH_CUSTOMER_LIC_PORT_CM3 TRUE
- #define CH_CUSTOMER_LIC_PORT_CM4 TRUE
- #define CH_CUSTOMER_LIC_PORT_CM7 TRUE
- #define CH_CUSTOMER_LIC_PORT_CM33 TRUE
- #define CH_CUSTOMER_LIC_PORT_ARM79 TRUE
- #define CH_CUSTOMER_LIC_PORT_E200Z0 TRUE
- #define CH_CUSTOMER_LIC_PORT_E200Z2 TRUE
- #define CH_CUSTOMER_LIC_PORT_E200Z3 TRUE
- #define CH_CUSTOMER_LIC_PORT_E200Z4 TRUE

8.4.1 Detailed Description

Customer-related info.

8.5 chlicense.h File Reference

License Module macros and structures.

```
#include "chversion.h"
#include "chcustomer.h"
#include "chpartner.h"
#include "chruntime.h"
```

Macros

- #define CH_LICENSE_TYPE_STRING "GNU General Public License 3 (GPL3)"

  License identification string.
- #define CH_LICENSE_ID_STRING "N/A"

  Customer identification string.
- #define CH_LICENSE_ID_CODE "N/A"

  Customer code.
- #define CH_LICENSE_MODIFIABLE_CODE TRUE

  Code modifiability restrictions.
- #define CH_LICENSE_FEATURES CH_FEATURES_FULL

  Code functionality restrictions.
- #define CH_LICENSE_MAX_DEPLOY CH_DEPLOY_UNLIMITED

  Code deploy restrictions.

Allowed Features Levels

- #define CH_FEATURES_BASIC 0
- #define CH_FEATURES_INTERMEDIATE 1
- #define CH_FEATURES_FULL 2

Deployment Options

- #define CH_DEPLOY_UNLIMITED -1
- #define CH_DEPLOY_NONE 0

Licensing Options

- #define CH_LICENSE_GPL 0
- #define CH_LICENSE_GPL_EXCEPTION 1
- #define CH_LICENSE_COMMERCIAL_FREE 2
- #define CH_LICENSE_COMMERCIAL_DEV_1000 3
- #define CH_LICENSE_COMMERCIAL_DEV_5000 4
- #define CH_LICENSE_COMMERCIAL_FULL 5
- #define CH_LICENSE_COMMERCIAL_RUNTIME 6
- #define CH_LICENSE_PARTNER 7

8.5.1 Detailed Description

License Module macros and structures.

8.6 chversion.h File Reference

Version Module macros and structures.

Macros

- #define __CHIBIOS__
  ChibiOS product identification macro.
- #define CH_VERSION_STABLE 1
  Stable release flag.
- #define CH_VERSION_DATE (((CH_VERSION_YEAR + 2000) * 100) + CH_VERSION_MONTH)
  Current version date in numeric form (yyyymm).

ChibiOS version identification

- #define CH_VERSION "2021.11.4"
  ChibiOS version string.
- #define CH_VERSION_YEAR 21
  ChibiOS version release year.
- #define CH_VERSION_MONTH 11
  ChibiOS version release month.
- #define CH_VERSION_PATCH 4
  ChibiOS version patch number.
- #define CH_VERSION_NICKNAME "Paestum"
  ChibiOS version nickname.

8.6.1 Detailed Description

Version Module macros and structures.

8.7 license.dox File Reference

8.8 lib.dox File Reference

8.9 chbsem.h File Reference

Binary semaphores structures and macros.

Data Structures

- struct ch_binary_semaphore
  Binary semaphore type.

Macros

- #define __BSEMAPHORE_DATA(name, taken)

  Data part of a static semaphore initializer.
- #define BSEMAPHORE_DECL(name, taken)

  Static semaphore initializer.

Typedefs

- typedef struct ch_binary_semaphore binary_semaphore_t

  Binary semaphore type.

Functions

- static void chBSemObjectInit (binary_semaphore_t *bsp, bool taken)

  Initializes a binary semaphore.
- static msg_t chBSemWait (binary_semaphore_t *bsp)

  Wait operation on the binary semaphore.
- static msg_t chBSemWaitS (binary_semaphore_t *bsp)

  Wait operation on the binary semaphore.
- static msg_t chBSemWaitTimeoutS (binary_semaphore_t *bsp, sysinterval_t timeout)

  Wait operation on the binary semaphore.
- static msg_t chBSemWaitTimeout (binary_semaphore_t *bsp, sysinterval_t timeout)

  Wait operation on the binary semaphore.
- static void chBSemResetI (binary_semaphore_t *bsp, bool taken)

  Reset operation on the binary semaphore.
- static void chBSemReset (binary_semaphore_t *bsp, bool taken)

  Reset operation on the binary semaphore.
- static void chBSemSignalI (binary_semaphore_t *bsp)

  Performs a signal operation on a binary semaphore.
- static void chBSemSignal (binary_semaphore_t *bsp)

  Performs a signal operation on a binary semaphore.
- static bool chBSemGetStateI (const binary_semaphore_t *bsp)

  Returns the binary semaphore current state.

8.9.1 Detailed Description

Binary semaphores structures and macros.

Binary semaphores related APIs and services.

Operation mode

Binary semaphores are implemented as a set of inline functions that use the existing counting semaphores primitives. The difference between counting and binary semaphores is that the counter of binary semaphores is not allowed to grow above the value 1. Repeated signal operation are ignored. A binary semaphore can thus have only two defined states:

- **Taken**, when its counter has a value of zero or lower than zero. A negative number represent the number of threads queued on the binary semaphore.
- **Not taken**, when its counter has a value of one.

Binary semaphores are different from mutexes because there is no concept of ownership, a binary semaphore can be taken by a thread and signaled by another thread or an interrupt handler, mutexes can only be taken and released by the same thread. Another difference is that binary semaphores, unlike mutexes, do not implement the priority inheritance protocol.
In order to use the binary semaphores APIs the `CH_CFG_USE_SEMAPHORES` option must be enabled in `chconf.h`.

8.10 chdelegates.h File Reference

Delegate threads macros and structures.

```
#include <stdarg.h>
```

Typedefs

- typedef msg_t(∗ delegate_veneer_t) (va_list ∗argsp)
  Type of a delegate veneer function.
- typedef msg_t(∗ delegate_fn0_t) (void)
  Type of a delegate function with no parameters.
- typedef msg_t(∗ delegate_fn1_t) (msg_t p1)
  Type of a delegate function with one parameter.
- typedef msg_t(∗ delegate_fn2_t) (msg_t p1, msg_t p2)
  Type of a delegate function with two parameters.
- typedef msg_t(∗ delegate_fn3_t) (msg_t p1, msg_t p2, msg_t p3)
  Type of a delegate function with three parameters.
- typedef msg_t(∗ delegate_fn4_t) (msg_t p1, msg_t p2, msg_t p3, msg_t p4)
  Type of a delegate function with four parameters.

Functions

- msg_t __ch_delegate_fn0 (va_list *argsp)
  Veneer for functions with no parameters.
- msg_t __ch_delegate_fn1 (va_list *argsp)
  Veneer for functions with one parameter.
- msg_t __ch_delegate_fn2 (va_list *argsp)
  Veneer for functions with two parameters.
- msg_t __ch_delegate_fn3 (va_list *argsp)
  Veneer for functions with three parameters.
- msg_t __ch_delegate_fn4 (va_list *argsp)
  Veneer for functions with four parameters.
- void chDelegateDispatch (void)
  Call messages dispatching.
- msg_t chDelegateDispatchTimeout (sysinterval_t timeout)
  Call messages dispatching with timeout.
- msg_t chDelegateCallVeneer (thread_t *tp, delegate_veneer_t veneer,...)
  Triggers a function call on a delegate thread.
- static msg_t chDelegateCallDirect0 (thread_t *tp, delegate_fn0_t func)
  Direct call to a function with no parameters.
- static msg_t chDelegateCallDirect1 (thread_t *tp, delegate_fn1_t func, msg_t p1)
  Direct call to a function with one parameter.
- static msg_t chDelegateCallDirect2 (thread_t *tp, delegate_fn2_t func, msg_t p1, msg_t p2)
  Direct call to a function with two parameters.
- static msg_t chDelegateCallDirect3 (thread_t *tp, delegate_fn3_t func, msg_t p1, msg_t p2, msg_t p3)
  Direct call to a function with three parameters.
- static msg_t chDelegateCallDirect4 (thread_t *tp, delegate_fn4_t func, msg_t p1, msg_t p2, msg_t p3, msg_t p4)
  Direct call to a function with four parameters.

8.10.1 Detailed Description

Delegate threads macros and structures.

8.11 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.
- #define CH_FACTORY_REQUIRES_POOLS
- #define CH_FACTORY_REQUIRES_HEAP

Typedefs

- typedef struct ch_dyn_element dyn_element_t

  Type of a dynamic object list element.
- typedef struct ch_dyn_list dyn_list_t

  Type of a dynamic object list.
- typedef struct ch_registered_static_object registered_object_t

  Type of a registered object.
- typedef struct ch_dyn_object dyn_buffer_t

Type of a dynamic buffer object.

- typedef struct ch_dyn_semaphore dyn_semaphore_t

  Type of a dynamic semaphore.
- typedef struct ch_dyn_mailbox dyn_mailbox_t

  Type of a dynamic buffer object.
- typedef struct ch_dyn_objects_fifo dyn_objects_fifo_t

  Type of a dynamic buffer object.
- typedef struct ch_dyn_pipe dyn_pipe_t

  Type of a dynamic pipe object.
- typedef struct ch_objects_factory objects_factory_t

  Type of the factory main object.

Functions

- void __factory_init (void)

  Initializes the objects factory.
- registered_object_t * chFactoryRegisterObject (const char *name, void *objp)

  Registers a generic object.
- registered_object_t * chFactoryFindObject (const char *name)

  Retrieves a registered object.
- registered_object_t * chFactoryFindObjectByPointer (void *objp)

  Retrieves a registered object by pointer.
- void chFactoryReleaseObject (registered_object_t *rop)

  Releases a registered object.
- dyn_buffer_t * chFactoryCreateBuffer (const char *name, size_t size)

  Creates a generic dynamic buffer object.
- dyn_buffer_t * chFactoryFindBuffer (const char *name)

  Retrieves a dynamic buffer object.
- void chFactoryReleaseBuffer (dyn_buffer_t *dbp)

  Releases a dynamic buffer object.
- dyn_semaphore_t * chFactoryCreateSemaphore (const char *name, cnt_t n)

  Creates a dynamic semaphore object.
- dyn_semaphore_t * chFactoryFindSemaphore (const char *name)

  Retrieves a dynamic semaphore object.
- void chFactoryReleaseSemaphore (dyn_semaphore_t *dsp)

  Releases a dynamic semaphore object.
- dyn_mailbox_t * chFactoryCreateMailbox (const char *name, size_t n)

  Creates a dynamic mailbox object.
- dyn_mailbox_t * chFactoryFindMailbox (const char *name)

  Retrieves a dynamic mailbox object.
- void chFactoryReleaseMailbox (dyn_mailbox_t *dmp)

  Releases a dynamic mailbox object.
- 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.

8.11.1 Detailed Description

ChibiOS objects factory structures and macros.

8.12 chjobs.h File Reference

Jobs Queues structures and macros.

Data Structures

- struct ch_jobs_queue

  Type of a jobs queue.
- struct ch_job_descriptor

  Type of a job descriptor.

Macros

- #define MSG_JOB_NULL ((msg_t)-2)

  Dispatcher return code in case of a `JOB_NUL` has been received.

Typedefs

- typedef struct ch_jobs_queue jobs_queue_t
  Type of a jobs queue.
- typedef void(* job_function_t) (void *arg)
  Type of a job function.
- typedef struct ch_job_descriptor job_descriptor_t
  Type of a job descriptor.

Functions

- static void chJobObjectInit (jobs_queue_t *jqp, size_t jobsn, job_descriptor_t *jobsbuf, msg_t *msgbuf)
  Initializes a jobs queue object.
- static job_descriptor_t * chJobGet (jobs_queue_t *jqp)
  Allocates a free job object.
- static job_descriptor_t * chJobGetI (jobs_queue_t *jqp)
  Allocates a free job object.
- static job_descriptor_t * chJobGetTimeoutS (jobs_queue_t *jqp, sysinterval_t timeout)
  Allocates a free job object.
- static job_descriptor_t * chJobGetTimeout (jobs_queue_t *jqp, sysinterval_t timeout)
  Allocates a free job object.
- static void chJobPostI (jobs_queue_t *jqp, job_descriptor_t *jp)
  Posts a job object.
- static void chJobPostS (jobs_queue_t *jqp, job_descriptor_t *jp)
  Posts a job object.
- static void chJobPost (jobs_queue_t *jqp, job_descriptor_t *jp)
  Posts a job object.
- static void chJobPostAheadI (jobs_queue_t *jqp, job_descriptor_t *jp)
  Posts an high priority job object.
- static void chJobPostAheadS (jobs_queue_t *jqp, job_descriptor_t *jp)
  Posts an high priority job object.
- static void chJobPostAhead (jobs_queue_t *jqp, job_descriptor_t *jp)
  Posts an high priority job object.
- static msg_t chJobDispatch (jobs_queue_t *jqp)
  Waits for a job then executes it.
- static msg_t chJobDispatchTimeout (jobs_queue_t *jqp, sysinterval_t timeout)
  Waits for a job then executes it.

8.12.1 Detailed Description

Jobs Queues structures and macros.

This module implements queues of generic jobs to be delegated asynchronously to a pool of dedicated threads. Operations defined for Jobs Queues

- **Get**: An job object is taken from the pool of the available jobs.
- **Post**: A job is posted to the queue, it will be returned to the pool after execution.

8.13 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 "chobjcaches.h"
#include "chdelegates.h"
#include "chjobs.h"
#include "chfactory.h"
```

Macros

- #define __CHIBIOS_OSLIB__

  ChibiOS/LIB identification macro.
- #define CH_OSLIB_STABLE 1

  Stable release flag.
- #define CH_CFG_USE_FACTORY FALSE
- #define CH_CFG_USE_HEAP FALSE
- #define CH_CFG_USE_MEMPOOLS FALSE
- #define CH_CFG_USE_OBJ_FIFOS FALSE
- #define CH_CFG_USE_PIPES FALSE
- #define CH_CFG_USE_OBJ_CACHES FALSE
- #define CH_CFG_USE_DELEGATES FALSE
- #define CH_CFG_USE_JOBS FALSE
- #define CH_CFG_USE_MAILBOXES FALSE

ChibiOS/LIB version identification

- #define CH_OSLIB_VERSION "1.3.2"

  OS Library version string.
- #define CH_OSLIB_MAJOR 1

  OS Library version major number.
- #define CH_OSLIB_MINOR 3

  OS Library version minor number.
- #define CH_OSLIB_PATCH 2

  OS Library version patch number.

Functions

- static void __oslib_init (void)

  Initialization of all library modules.

8.13.1 Detailed Description

ChibiOS/LIB main include file.

This header includes all the required library headers. This file is meant do be included by ch.h not directly by user.

8.14 chmboxes.h File Reference

Mailboxes macros and structures.

Data Structures

- struct mailbox_t

  Structure representing a mailbox object.

Macros

- #define __MAILBOX_DATA(name, buffer, size)

  Data part of a static mailbox initializer.
- #define MAILBOX_DECL(name, buffer, size)

  Static mailbox initializer.

Functions

- void chMBObjectInit (mailbox_t *mbp, msg_t *buf, size_t n)

  Initializes a mailbox_t object.
- void chMBReset (mailbox_t *mbp)

  Resets a mailbox_t object.
- void chMBResetI (mailbox_t *mbp)

  Resets a mailbox_t object.
- msg_t chMBPostTimeout (mailbox_t *mbp, msg_t msg, sysinterval_t timeout)

  Posts a message into a mailbox.
- msg_t chMBPostTimeoutS (mailbox_t *mbp, msg_t msg, sysinterval_t timeout)

  Posts a message into a mailbox.
- msg_t chMBPostI (mailbox_t *mbp, msg_t msg)

  Posts a message into a mailbox.
- msg_t chMBPostAheadTimeout (mailbox_t *mbp, msg_t msg, sysinterval_t timeout)

  Posts an high priority message into a mailbox.
- msg_t chMBPostAheadTimeoutS (mailbox_t *mbp, msg_t msg, sysinterval_t timeout)

  Posts an high priority message into a mailbox.
- msg_t chMBPostAheadI (mailbox_t *mbp, msg_t msg)

  Posts an high priority message into a mailbox.
- msg_t chMBFetchTimeout (mailbox_t *mbp, msg_t *msgp, sysinterval_t timeout)

  Retrieves a message from a mailbox.
- msg_t chMBFetchTimeoutS (mailbox_t *mbp, msg_t *msgp, sysinterval_t timeout)

  Retrieves a message from a mailbox.

- msg_t chMBFetchI (mailbox_t *mbp, msg_t *msgp)
  Retrieves a message from a mailbox.
- static size_t chMBGetSizeI (const mailbox_t *mbp)
  Returns the mailbox buffer size as number of messages.
- static size_t chMBGetUsedCountI (const mailbox_t *mbp)
  Returns the number of used message slots into a mailbox.
- static size_t chMBGetFreeCountI (const mailbox_t *mbp)
  Returns the number of free message slots into a mailbox.
- static msg_t chMBPeekI (const mailbox_t *mbp)
  Returns the next message in the queue without removing it.
- static void chMBResumeX (mailbox_t *mbp)
  Terminates the reset state.

8.14.1 Detailed Description

Mailboxes macros and structures.

8.15 chmemcore.h File Reference

Core memory manager macros and structures.

Data Structures

- struct memcore_t
  Type of memory core object.

Macros

- #define CH_CFG_MEMCORE_SIZE 0
  Managed RAM size.
- #define chCoreAllocAlignedWithOffsetI chCoreAllocFromTopI
  Allocates a memory block.
- #define chCoreAllocAlignedWithOffset chCoreAllocFromTop
  Allocates a memory block.

Typedefs

- typedef void *(* memgetfunc_t) (size_t size, unsigned align)
  Memory get function.
- typedef void *(* memgetfunc2_t) (size_t size, unsigned align, size_t offset)
  Enhanced memory get function.

Functions

- void __core_init (void)

  Low level memory manager initialization.
- void * chCoreAllocFromBaseI (size_t size, unsigned align, size_t offset)

  Allocates a memory block starting from the lowest address upward.
- void * chCoreAllocFromTopI (size_t size, unsigned align, size_t offset)

  Allocates a memory block starting from the top address downward.
- void * chCoreAllocFromBase (size_t size, unsigned align, size_t offset)

  Allocates a memory block starting from the lowest address upward.
- void * chCoreAllocFromTop (size_t size, unsigned align, size_t offset)

  Allocates a memory block starting from the top address downward.
- size_t chCoreGetStatusX (void)

  Core memory status.
- static void * chCoreAllocAlignedI (size_t size, unsigned align)

  Allocates a memory block.
- static void * chCoreAllocAligned (size_t size, unsigned align)

  Allocates a memory block.
- static void * chCoreAllocI (size_t size)

  Allocates a memory block.
- static void * chCoreAlloc (size_t size)

  Allocates a memory block.

8.15.1 Detailed Description

Core memory manager macros and structures.

8.16 chmemheaps.h File Reference

Memory heaps macros and structures.

Data Structures

- union heap_header

  Memory heap block header.
- struct memory_heap

  Structure describing a memory heap.

Macros

- #define CH_HEAP_ALIGNMENT 8U

  Minimum alignment used for heap.
- #define CH_HEAP_AREA(name, size)

  Allocation of an aligned static heap buffer.

Typedefs

- typedef struct memory_heap memory_heap_t
  Type of a memory heap.
- typedef union heap_header heap_header_t
  Type of a memory heap header.

Functions

- void __heap_init (void)
  Initializes the default heap.
- void chHeapObjectInit (memory_heap_t *heapp, void *buf, size_t size)
  Initializes a memory heap from a static memory area.
- void * chHeapAllocAligned (memory_heap_t *heapp, size_t size, unsigned align)
  Allocates a block of memory from the heap by using the first-fit algorithm.
- void chHeapFree (void *p)
  Frees a previously allocated memory block.
- size_t chHeapStatus (memory_heap_t *heapp, size_t *totalp, size_t *largestp)
  Reports the heap status.
- static void * chHeapAlloc (memory_heap_t *heapp, size_t size)
  Allocates a block of memory from the heap by using the first-fit algorithm.
- static size_t chHeapGetSize (const void *p)
  Returns the size of an allocated block.

8.16.1 Detailed Description

Memory heaps macros and structures.

8.17 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)

  Data part of a static memory pool initializer.
- #define MEMORYPOOL_DECL(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)

  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 * chPoolAllocI (memory_pool_t *mp)

  Allocates an object from a memory pool.
- void * chPoolAlloc (memory_pool_t *mp)

  Allocates an object from a memory pool.
- void chPoolFreeI (memory_pool_t *mp, void *objp)

  Releases an object into a memory pool.
- void chPoolFree (memory_pool_t *mp, void *objp)

  Releases an object into a memory pool.
- void chGuardedPoolObjectInitAligned (guarded_memory_pool_t *gmp, size_t size, unsigned align)

  Initializes an empty guarded memory pool.
- void chGuardedPoolLoadArray (guarded_memory_pool_t *gmp, void *p, size_t n)

  Loads a guarded memory pool with an array of static objects.
- void * chGuardedPoolAllocTimeoutS (guarded_memory_pool_t *gmp, sysinterval_t timeout)

  Allocates an object from a guarded memory pool.
- void * chGuardedPoolAllocTimeout (guarded_memory_pool_t *gmp, sysinterval_t timeout)

  Allocates an object from a guarded memory pool.
- void chGuardedPoolFree (guarded_memory_pool_t *gmp, void *objp)

  Releases an object into a guarded memory pool.
- static void chPoolObjectInit (memory_pool_t *mp, size_t size, memgetfunc_t provider)

  Initializes an empty memory pool.
- static void chPoolAdd (memory_pool_t *mp, void *objp)

  Adds an object to a memory pool.
- static void chPoolAddI (memory_pool_t *mp, void *objp)

  Adds an object to a memory pool.
- static void chGuardedPoolObjectInit (guarded_memory_pool_t *gmp, size_t size)

  Initializes an empty guarded memory pool.
- static cnt_t chGuardedPoolGetCounterI (guarded_memory_pool_t *gmp)

  Gets the count of objects in a guarded memory pool.
- static void * chGuardedPoolAllocI (guarded_memory_pool_t *gmp)

  Allocates an object from a guarded memory pool.
- static void chGuardedPoolFreeI (guarded_memory_pool_t *gmp, void *objp)

  Releases an object into a guarded memory pool.
- static void chGuardedPoolFreeS (guarded_memory_pool_t *gmp, void *objp)

Releases an object into a guarded memory pool.

- static void chGuardedPoolAdd (guarded_memory_pool_t *gmp, void *objp)

  Adds an object to a guarded memory pool.
- static void chGuardedPoolAddI (guarded_memory_pool_t *gmp, void *objp)

  Adds an object to a guarded memory pool.
- static void chGuardedPoolAddS (guarded_memory_pool_t *gmp, void *objp)

  Adds an object to a guarded memory pool.

8.17.1 Detailed Description

Memory Pools macros and structures.

8.18 chobjcaches.h File Reference

Objects Caches macros and structures.

Data Structures

- struct ch_oc_hash_header

  Structure representing an hash table element.
- struct ch_oc_lru_header

  Structure representing an hash table element.
- struct ch_oc_object

  Structure representing a cached object.
- struct ch_objects_cache

  Structure representing a cache object.

Macros

Cached objects flags

- #define OC_FLAG_INLRU 0x00000001U
- #define OC_FLAG_INHASH 0x00000002U
- #define OC_FLAG_SHARED 0x00000004U
- #define OC_FLAG_NOTSYNC 0x00000008U
- #define OC_FLAG_LAZYWRITE 0x00000010U
- #define OC_FLAG_FORGET 0x00000020U

Typedefs

- typedef uint32_t oc_flags_t

  Flags of cached objects.
- typedef struct ch_oc_hash_header oc_hash_header_t

  Type of an hash element header.
- typedef struct ch_oc_lru_header oc_lru_header_t

  Type of an LRU element header.
- typedef struct ch_oc_object oc_object_t

  Type of a cached object.
- typedef struct ch_objects_cache objects_cache_t

  Type of a cache object.
- typedef bool(* oc_readf_t) (objects_cache_t *ocp, oc_object_t *objp, bool async)

  Object read function.
- typedef bool(* oc_writef_t) (objects_cache_t *ocp, oc_object_t *objp, bool async)

  Object write function.

Functions

- void chCacheObjectInit (objects_cache_t *ocp, ucnt_t hashn, oc_hash_header_t *hashp, ucnt_t objn, size_t objsz, void *objvp, oc_readf_t readf, oc_writef_t writef)

  Initializes a `objects_cache_t` object.
- oc_object_t * chCacheGetObject (objects_cache_t *ocp, uint32_t group, uint32_t key)

  Retrieves an object from the cache.
- void chCacheReleaseObjectI (objects_cache_t *ocp, oc_object_t *objp)

  Releases an object into the cache.
- bool chCacheReadObject (objects_cache_t *ocp, oc_object_t *objp, bool async)

  Reads object data from the storage.
- bool chCacheWriteObject (objects_cache_t *ocp, oc_object_t *objp, bool async)

  Writes the object data back to storage.
- static void chCacheReleaseObject (objects_cache_t *ocp, oc_object_t *objp)

  Releases an object into the cache.

8.18.1 Detailed Description

Objects Caches macros and structures.

8.19 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.

8.19.1 Detailed Description

Objects FIFO structures and macros.

This module implements a generic FIFO queue of objects by coupling a Guarded Memory Pool (for objects storage) and a MailBox.
On the sender side free objects are taken from the pool, filled and then sent to the receiver, on the receiver side objects are fetched, used and then returned to the pool. Operations defined for object FIFOs:

- **Take**: An object is taken from the pool of the free objects, can be blocking.
- **Return**: An object is returned to the pool of the free objects, it is guaranteed to be non-blocking.
- **Send**: An object is sent through the mailbox, it is guaranteed to be non-blocking
- **Receive**: An object is received from the mailbox, can be blocking.

8.20 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)

  Static pipe initializer.

Functions

- void chPipeObjectInit (pipe_t *pp, uint8_t *buf, size_t n)

  Initializes a `mailbox_t` object.
- void chPipeReset (pipe_t *pp)

  Resets a `pipe_t` object.
- size_t chPipeWriteTimeout (pipe_t *pp, const uint8_t *bp, size_t n, sysinterval_t timeout)

  Pipe write with timeout.
- size_t chPipeReadTimeout (pipe_t *pp, uint8_t *bp, size_t n, sysinterval_t timeout)

  Pipe read with timeout.
- static size_t chPipeGetSize (const pipe_t *pp)

  Returns the pipe buffer size as number of bytes.
- static size_t chPipeGetUsedCount (const pipe_t *pp)

  Returns the number of used byte slots into a pipe.
- static size_t chPipeGetFreeCount (const pipe_t *pp)

  Returns the number of free byte slots into a pipe.
- static void chPipeResume (pipe_t *pp)

  Terminates the reset state.

8.20.1 Detailed Description

Pipes macros and structures.

8.21 chdelegates.c File Reference

Delegate threads code.

```
#include "ch.h"
```

Functions

- msg_t __ch_delegate_fn0 (va_list *argsp)
  Veneer for functions with no parameters.
- msg_t __ch_delegate_fn1 (va_list *argsp)
  Veneer for functions with one parameter.
- msg_t __ch_delegate_fn2 (va_list *argsp)
  Veneer for functions with two parameters.
- msg_t __ch_delegate_fn3 (va_list *argsp)
  Veneer for functions with three parameters.
- msg_t __ch_delegate_fn4 (va_list *argsp)
  Veneer for functions with four parameters.
- msg_t chDelegateCallVeneer (thread_t *tp, delegate_veneer_t veneer,...)
  Triggers a function call on a delegate thread.
- void chDelegateDispatch (void)
  Call messages dispatching.
- msg_t chDelegateDispatchTimeout (sysinterval_t timeout)
  Call messages dispatching with timeout.

8.21.1 Detailed Description

Delegate threads code.

Delegate threads.

Operation mode

A delegate thread is a thread performing function calls triggered by other threads. This functionality is especially useful when encapsulating a library not designed for threading into a delegate thread. Other threads have access to the library without having to worry about mutual exclusion.

Precondition

In order to use the pipes APIs the `CH_CFG_USE_DELEGATES` option must be enabled in `chconf.h`.

Note

Compatible with RT and NIL.

8.22 chfactory.c File Reference

ChibiOS objects factory and registry code.

```
#include <string.h>
#include "ch.h"
```

Macros

- #define F_LOCK()
- #define F_UNLOCK()

Functions

- static void copy_name (const char *sp, char *dp)
- static void dyn_list_init (dyn_list_t *dlp)
- static dyn_element_t * dyn_list_find (const char *name, dyn_list_t *dlp)
- static dyn_element_t * dyn_list_find_prev (dyn_element_t *element, dyn_list_t *dlp)
- static dyn_element_t * dyn_list_unlink (dyn_element_t *prev)
- static dyn_element_t * dyn_create_object_heap (const char *name, dyn_list_t *dlp, size_t size, unsigned align)
- static void dyn_release_object_heap (dyn_element_t *dep, dyn_list_t *dlp)
- static dyn_element_t * dyn_create_object_pool (const char *name, dyn_list_t *dlp, memory_pool_t *mp)
- static void dyn_release_object_pool (dyn_element_t *dep, dyn_list_t *dlp, memory_pool_t *mp)
- static dyn_element_t * dyn_find_object (const char *name, dyn_list_t *dlp)
- void __factory_init (void)

  Initializes the objects factory.
- registered_object_t * chFactoryRegisterObject (const char *name, void *objp)

  Registers a generic object.
- registered_object_t * chFactoryFindObject (const char *name)

  Retrieves a registered object.
- registered_object_t * chFactoryFindObjectByPointer (void *objp)

  Retrieves a registered object by pointer.
- void chFactoryReleaseObject (registered_object_t *rop)

  Releases a registered object.
- dyn_buffer_t * chFactoryCreateBuffer (const char *name, size_t size)

  Creates a generic dynamic buffer object.
- dyn_buffer_t * chFactoryFindBuffer (const char *name)

  Retrieves a dynamic buffer object.
- void chFactoryReleaseBuffer (dyn_buffer_t *dbp)

  Releases a dynamic buffer object.
- dyn_semaphore_t * chFactoryCreateSemaphore (const char *name, cnt_t n)

  Creates a dynamic semaphore object.
- dyn_semaphore_t * chFactoryFindSemaphore (const char *name)

  Retrieves a dynamic semaphore object.
- void chFactoryReleaseSemaphore (dyn_semaphore_t *dsp)

  Releases a dynamic semaphore object.
- dyn_mailbox_t * chFactoryCreateMailbox (const char *name, size_t n)

  Creates a dynamic mailbox object.
- dyn_mailbox_t * chFactoryFindMailbox (const char *name)

  Retrieves a dynamic mailbox object.
- void chFactoryReleaseMailbox (dyn_mailbox_t *dmp)

  Releases a dynamic mailbox object.
- dyn_objects_fifo_t * chFactoryCreateObjectsFIFO (const char *name, size_t objsize, size_t objn, unsigned objalign)

  Creates a dynamic "objects FIFO" object.
- dyn_objects_fifo_t * chFactoryFindObjectsFIFO (const char *name)

  Retrieves a dynamic "objects FIFO" object.

- void chFactoryReleaseObjectsFIFO (dyn_objects_fifo_t *dofp)
  Releases a dynamic "objects FIFO" object.
- dyn_pipe_t * chFactoryCreatePipe (const char *name, size_t size)
  Creates a dynamic pipe object.
- dyn_pipe_t * chFactoryFindPipe (const char *name)
  Retrieves a dynamic pipe object.
- void chFactoryReleasePipe (dyn_pipe_t *dpp)
  Releases a dynamic pipe object.

Variables

- objects_factory_t ch_factory
  Factory object static instance.

8.22.1 Detailed Description

ChibiOS objects factory and registry code.

8.23 chmboxes.c File Reference

Mailboxes code.

```
#include "ch.h"
```

Functions

- void chMBObjectInit (mailbox_t *mbp, msg_t *buf, size_t n)
  Initializes a `mailbox_t` object.
- void chMBReset (mailbox_t *mbp)
  Resets a `mailbox_t` object.
- void chMBResetI (mailbox_t *mbp)
  Resets a `mailbox_t` object.
- msg_t chMBPostTimeout (mailbox_t *mbp, msg_t msg, sysinterval_t timeout)
  Posts a message into a mailbox.
- msg_t chMBPostTimeoutS (mailbox_t *mbp, msg_t msg, sysinterval_t timeout)
  Posts a message into a mailbox.
- msg_t chMBPostI (mailbox_t *mbp, msg_t msg)
  Posts a message into a mailbox.
- msg_t chMBPostAheadTimeout (mailbox_t *mbp, msg_t msg, sysinterval_t timeout)
  Posts an high priority message into a mailbox.
- msg_t chMBPostAheadTimeoutS (mailbox_t *mbp, msg_t msg, sysinterval_t timeout)
  Posts an high priority message into a mailbox.
- msg_t chMBPostAheadI (mailbox_t *mbp, msg_t msg)
  Posts an high priority message into a mailbox.
- msg_t chMBFetchTimeout (mailbox_t *mbp, msg_t *msgp, sysinterval_t timeout)
  Retrieves a message from a mailbox.
- msg_t chMBFetchTimeoutS (mailbox_t *mbp, msg_t *msgp, sysinterval_t timeout)
  Retrieves a message from a mailbox.
- msg_t chMBFetchI (mailbox_t *mbp, msg_t *msgp)
  Retrieves a message from a mailbox.

8.23.1 Detailed Description

Mailboxes code.

8.24 chmemcore.c File Reference

Core memory manager code.

```
#include "ch.h"
```

Functions

- void __core_init (void)
  Low level memory manager initialization.
- void * chCoreAllocFromBaseI (size_t size, unsigned align, size_t offset)
  Allocates a memory block starting from the lowest address upward.
- void * chCoreAllocFromTopI (size_t size, unsigned align, size_t offset)
  Allocates a memory block starting from the top address downward.
- void * chCoreAllocFromBase (size_t size, unsigned align, size_t offset)
  Allocates a memory block starting from the lowest address upward.
- void * chCoreAllocFromTop (size_t size, unsigned align, size_t offset)
  Allocates a memory block starting from the top address downward.
- size_t chCoreGetStatusX (void)
  Core memory status.

Variables

- memcore_t ch_memcore
  Memory core descriptor.

8.24.1 Detailed Description

Core memory manager code.

8.25 chmemheaps.c File Reference

Memory heaps code.

```
#include "ch.h"
```

Macros

- #define H_LOCK(h)
- #define H_UNLOCK(h)
- #define H_BLOCK(hp)
- #define H_LIMIT(hp)
- #define H_NEXT(hp)
- #define H_PAGES(hp)
- #define H_HEAP(hp)
- #define H_SIZE(hp)
- #define NPAGES(p1, p2)

Functions

- void __heap_init (void)

  Initializes the default heap.
- void chHeapObjectInit (memory_heap_t *heapp, void *buf, size_t size)

  Initializes a memory heap from a static memory area.
- void * chHeapAllocAligned (memory_heap_t *heapp, size_t size, unsigned align)

  Allocates a block of memory from the heap by using the first-fit algorithm.
- void chHeapFree (void *p)

  Frees a previously allocated memory block.
- size_t chHeapStatus (memory_heap_t *heapp, size_t *totalp, size_t *largestp)

  Reports the heap status.

Variables

- static memory_heap_t default_heap

  Default heap descriptor.

8.25.1 Detailed Description

Memory heaps code.

8.26 chmempools.c File Reference

Memory Pools code.

```
#include "ch.h"
```

Functions

- void chPoolObjectInitAligned (memory_pool_t *mp, size_t size, unsigned align, memgetfunc_t provider)
  Initializes an empty memory pool.
- void chPoolLoadArray (memory_pool_t *mp, void *p, size_t n)
  Loads a memory pool with an array of static objects.
- void * chPoolAllocI (memory_pool_t *mp)
  Allocates an object from a memory pool.
- void * chPoolAlloc (memory_pool_t *mp)
  Allocates an object from a memory pool.
- void chPoolFreeI (memory_pool_t *mp, void *objp)
  Releases an object into a memory pool.
- void chPoolFree (memory_pool_t *mp, void *objp)
  Releases an object into a memory pool.
- void chGuardedPoolObjectInitAligned (guarded_memory_pool_t *gmp, size_t size, unsigned align)
  Initializes an empty guarded memory pool.
- void chGuardedPoolLoadArray (guarded_memory_pool_t *gmp, void *p, size_t n)
  Loads a guarded memory pool with an array of static objects.
- void * chGuardedPoolAllocTimeoutS (guarded_memory_pool_t *gmp, sysinterval_t timeout)
  Allocates an object from a guarded memory pool.
- void * chGuardedPoolAllocTimeout (guarded_memory_pool_t *gmp, sysinterval_t timeout)
  Allocates an object from a guarded memory pool.
- void chGuardedPoolFree (guarded_memory_pool_t *gmp, void *objp)
  Releases an object into a guarded memory pool.

8.26.1 Detailed Description

Memory Pools code.

8.27 chobjcaches.c File Reference

Objects Caches code.

```
#include "ch.h"
```

Macros

- #define OC_HASH_FUNCTION(ocp, group, key)
- #define HASH_INSERT(ocp, objp, group, key)
- #define HASH_REMOVE(objp)
- #define LRU_INSERT_HEAD(ocp, objp)
- #define LRU_INSERT_TAIL(ocp, objp)
- #define LRU_REMOVE(objp)

Functions

- static oc_object_t * hash_get_s (objects_cache_t *ocp, uint32_t group, uint32_t key)

  Returns an object pointer from the cache, if present.
- static oc_object_t * lru_get_last_s (objects_cache_t *ocp)

  Gets the least recently used object buffer from the LRU list.
- void chCacheObjectInit (objects_cache_t *ocp, ucnt_t hashn, oc_hash_header_t *hashp, ucnt_t objn, size_t objsz, void *objvp, oc_readf_t readf, oc_writef_t writef)

  Initializes a `objects_cache_t` object.
- oc_object_t * chCacheGetObject (objects_cache_t *ocp, uint32_t group, uint32_t key)

  Retrieves an object from the cache.
- void chCacheReleaseObjectI (objects_cache_t *ocp, oc_object_t *objp)

  Releases an object into the cache.
- bool chCacheReadObject (objects_cache_t *ocp, oc_object_t *objp, bool async)

  Reads object data from the storage.
- bool chCacheWriteObject (objects_cache_t *ocp, oc_object_t *objp, bool async)

  Writes the object data back to storage.

8.27.1 Detailed Description

Objects Caches code.

Objects caches.

Operation mode

An object cache allows to retrieve and release objects from a slow media, for example a disk or flash.
The most recently used objects are kept in a series of RAM buffers making access faster. Objects are identified by a pair <group, key> which could be mapped, for example, to a disk drive identifier and sector identifier.
Read and write operations are performed using externally-supplied functions, the cache is device-agnostic.
The cache uses internally an hash table, the size of the table should be dimensioned to minimize the risk of hash collisions, a factor of two is usually acceptable, it depends on the specific application requirements.
Operations defined for caches:

- **Get Object**: Retrieves an object from cache, if not present then an empty buffer is returned.
- **Read Object**: Retrieves an object from cache, if not present a buffer is allocated and the object is read from the media.
- **Release Object**: Releases an object to the cache handling the media update, if required.

Precondition

In order to use the pipes APIs the `CH_CFG_USE_OBJ_CACHES` option must be enabled in `chconf.h`.

Note

Compatible with RT and NIL.

8.28 chpipes.c File Reference

Pipes code.

```
#include <string.h>
#include "ch.h"
```

Macros

- #define PC_INIT(p)
- #define PC_LOCK(p)
- #define PC_UNLOCK(p)
- #define PW_INIT(p)
- #define PW_LOCK(p)
- #define PW_UNLOCK(p)
- #define PR_INIT(p)
- #define PR_LOCK(p)
- #define PR_UNLOCK(p)

Functions

- static size_t pipe_write (pipe_t *pp, const uint8_t *bp, size_t n)

  Non-blocking pipe write.
- static size_t pipe_read (pipe_t *pp, uint8_t *bp, size_t n)

  Non-blocking pipe read.
- void chPipeObjectInit (pipe_t *pp, uint8_t *buf, size_t n)

  Initializes a `mailbox_t` object.
- void chPipeReset (pipe_t *pp)

  Resets a `pipe_t` object.
- size_t chPipeWriteTimeout (pipe_t *pp, const uint8_t *bp, size_t n, sysinterval_t timeout)

  Pipe write with timeout.
- size_t chPipeReadTimeout (pipe_t *pp, uint8_t *bp, size_t n, sysinterval_t timeout)

  Pipe read with timeout.

8.28.1 Detailed Description

Pipes code.

Byte pipes.

Operation mode

A pipe is an asynchronous communication mechanism.
Operations defined for mailboxes:

- **Write**: Writes a buffer of data in the pipe in FIFO order.
- **Read**: A buffer of data is read from the read and removed.
- **Reset**: The pipe is emptied and all the stored data is lost.

Precondition

In order to use the pipes APIs the `CH_CFG_USE_PIPES` option must be enabled in `chconf.h`.

Note

Compatible with RT and NIL.

8.29 rt.dox File Reference

8.30 rt_arch.dox File Reference

8.31 ch.h File Reference

ChibiOS/RT main include file.

```
#include "chlicense.h"
#include "chconf.h"
#include "chchecks.h"
#include "chrestrictions.h"
#include "chearly.h"
#include "chrfcu.h"
#include "chdebug.h"
#include "chtime.h"
#include "chlists.h"
#include "chalign.h"
#include "chtrace.h"
#include "chport.h"
#include "chtm.h"
#include "chstats.h"
#include "chobjects.h"
#include "chsys.h"
#include "chinstances.h"
#include "chvt.h"
#include "chschd.h"
#include "chthreads.h"
#include "chregistry.h"
#include "chsem.h"
#include "chmtx.h"
#include "chcond.h"
#include "chevents.h"
#include "chmsg.h"
#include "chlib.h"
#include "chdynamic.h"
```

Macros

- #define __CHIBIOS_RT__
  ChibiOS/RT identification macro.
- #define CH_KERNEL_STABLE 1
  Stable release flag.

ChibiOS/RT version identification

- #define CH_KERNEL_VERSION "7.0.5"
  Kernel version string.
- #define CH_KERNEL_MAJOR 7
  Kernel version major number.
- #define CH_KERNEL_MINOR 0
  Kernel version minor number.
- #define CH_KERNEL_PATCH 5
  Kernel version patch number.

Constants for configuration options

- #define FALSE 0
  Generic 'false' preprocessor boolean constant.
- #define TRUE 1
  Generic 'true' preprocessor boolean constant.

8.31.1 Detailed Description

ChibiOS/RT main include file.

8.32 chalign.h File Reference

Memory alignment macros and structures.

Macros

Memory alignment support macros

- #define MEM_ALIGN_MASK(a)
  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)
  Returns whatever a pointer or memory size is aligned.
- #define MEM_IS_VALID_ALIGNMENT(a)
  Returns whatever a constant is a valid alignment.

8.32.1 Detailed Description

Memory alignment macros and structures.

8.33 chchecks.h File Reference

Configuration file checks header.

8.33.1 Detailed Description

Configuration file checks header.

8.34 chcond.h File Reference

Condition Variables macros and structures.

Data Structures

- struct condition_variable

  condition_variable_t structure.

Macros

- #define __CONDVAR_DATA(name)

  Data part of a static condition variable initializer.
- #define CONDVAR_DECL(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 s `condition_variable_t` structure.
- void chCondSignal (condition_variable_t *cp)

  Signals one thread that is waiting on the condition variable.
- void chCondSignalI (condition_variable_t *cp)

  Signals one thread that is waiting on the condition variable.
- void chCondBroadcast (condition_variable_t *cp)

  Signals all threads that are waiting on the condition variable.
- void chCondBroadcastI (condition_variable_t *cp)

  Signals all threads that are waiting on the condition variable.
- msg_t chCondWait (condition_variable_t *cp)

  Waits on the condition variable releasing the mutex lock.
- msg_t chCondWaitS (condition_variable_t *cp)

  Waits on the condition variable releasing the mutex lock.
- msg_t chCondWaitTimeout (condition_variable_t *cp, sysinterval_t timeout)

  Waits on the condition variable releasing the mutex lock.
- msg_t chCondWaitTimeoutS (condition_variable_t *cp, sysinterval_t timeout)

  Waits on the condition variable releasing the mutex lock.

8.34.1 Detailed Description

Condition Variables macros and structures.

8.35 chdebug.h File Reference

Debug support macros and structures.

Data Structures

- struct ch_system_debug
  System debug data structure.

Macros

- #define __dbg_check_disable()
- #define __dbg_check_suspend()
- #define __dbg_check_enable()
- #define __dbg_check_lock()
- #define __dbg_check_unlock()
- #define __dbg_check_lock_from_isr()
- #define __dbg_check_unlock_from_isr()
- #define __dbg_check_enter_isr()
- #define __dbg_check_leave_isr()
- #define chDbgCheckClassI()
- #define chDbgCheckClassS()

Debug related settings

- #define CH_DBG_STACK_FILL_VALUE 0x55
  Fill value for thread stack area in debug mode.

Macro Functions

- #define chDbgCheck(c)
  Function parameters check.
- #define chDbgAssert(c, r)
  Condition assertion.

Typedefs

- typedef struct ch_system_debug system_debug_t
  System debug data structure.

Functions

- static void __dbg_object_init (system_debug_t *sdp)
  Debug support initialization.

8.35.1 Detailed Description

Debug support macros and structures.

8.36 chdynamic.h File Reference

Dynamic threads macros and structures.

Functions

- thread_t * chThdCreateFromHeap (memory_heap_t *heapp, size_t size, const char *name, tprio_t prio, tfunc_t pf, void *arg)

  Creates a new thread allocating the memory from the heap.
- thread_t * chThdCreateFromMemoryPool (memory_pool_t *mp, const char *name, tprio_t prio, tfunc_t pf, void *arg)

  Creates a new thread allocating the memory from the specified memory pool.

8.36.1 Detailed Description

Dynamic threads macros and structures.

8.37 chearly.h File Reference

Early forward types declarations header.

```
#include "chtypes.h"
```

Macros

- #define __CH_STRINGIFY(a)

  Utility to make the parameter a quoted string.
- #define __CH_OFFSETOF(st, m)

  Structure field offset utility.
- #define __CH_USED(x)

  Marks an expression result as used.
- #define likely(x)

  Marks a boolean expression as likely true.
- #define unlikely(x)

  Marks a boolean expression as likely false.
- #define threadref(p)

  Safe cast of a queue pointer to a thread pointer.

Typedefs

- typedef unsigned core_id_t

  Type of a core identifier.
- typedef struct ch_thread thread_t

  Type of a thread structure.
- typedef struct ch_os_instance os_instance_t

  Type of an OS instance structure.

Kernel types

- typedef port_rtcnt_t rtcnt_t
- typedef port_rttime_t rttime_t
- typedef port_syssts_t syssts_t
- typedef port_stkalign_t stkalign_t
- typedef uint8_t tmode_t
- typedef uint8_t tstate_t
- typedef uint8_t trefs_t
- typedef uint8_t tslices_t
- typedef uint32_t tprio_t
- typedef int32_t msg_t
- typedef int32_t eventid_t
- typedef uint32_t eventmask_t
- typedef uint32_t eventflags_t
- typedef int32_t cnt_t
- typedef uint32_t ucnt_t

Functions

- void chSysHalt (const char *reason)

  Halts the system.

8.37.1 Detailed Description

Early forward types declarations header.

8.38 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)

  Returns an event mask from an event identifier.
- #define __EVENTSOURCE_DATA(name)

  Data part of a static event source initializer.
- #define EVENTSOURCE_DECL(name)

  Static event source initializer.
- #define chEvtWaitOne(mask)
- #define chEvtWaitAny(mask)
- #define chEvtWaitAll(mask)

Typedefs

- typedef struct event_listener event_listener_t
- typedef struct event_source event_source_t

  Event Source structure.
- typedef void(∗ evhandler_t) (eventid_t id)

  Event Handler callback function.

Functions

- void chEvtRegisterMaskWithFlagsI (event_source_t ∗esp, event_listener_t ∗elp, eventmask_t events, eventflags_t wflags)

  Registers an Event Listener on an Event Source.
- void chEvtRegisterMaskWithFlags (event_source_t ∗esp, event_listener_t ∗elp, eventmask_t events, eventflags_t wflags)

  Registers an Event Listener on an Event Source.
- void chEvtUnregister (event_source_t ∗esp, event_listener_t ∗elp)

  Unregisters an Event Listener from its Event Source.
- eventmask_t chEvtGetAndClearEventsI (eventmask_t events)

  Clears the pending events specified in the events mask.
- eventmask_t chEvtGetAndClearEvents (eventmask_t events)

  Clears the pending events specified in the events mask.
- eventmask_t chEvtAddEvents (eventmask_t events)

  *Adds (OR) a set of events to the current thread, this is **much** faster than using `chEvtBroadcast()` or `chEvtSignal()`.*
- eventflags_t chEvtGetAndClearFlagsI (event_listener_t ∗elp)

  Returns the unmasked flags associated to an `event_listener_t`.
- eventflags_t chEvtGetAndClearFlags (event_listener_t ∗elp)

  Returns the flags associated to an `event_listener_t`.
- void chEvtSignal (thread_t ∗tp, eventmask_t events)

  Adds a set of event flags directly to the specified `thread_t`.
- void chEvtSignalI (thread_t ∗tp, eventmask_t events)

  Adds a set of event flags directly to the specified `thread_t`.
- void chEvtBroadcastFlags (event_source_t ∗esp, eventflags_t flags)

  Signals all the Event Listeners registered on the specified Event Source.
- void chEvtBroadcastFlagsI (event_source_t ∗esp, eventflags_t flags)

Signals all the Event Listeners registered on the specified Event Source.

- void chEvtDispatch (const evhandler_t *handlers, eventmask_t events)

  Invokes the event handlers associated to an event flags mask.
- eventmask_t chEvtWaitOne (eventmask_t events)

  Waits for exactly one of the specified events.
- eventmask_t chEvtWaitAny (eventmask_t events)

  Waits for any of the specified events.
- eventmask_t chEvtWaitAll (eventmask_t events)

  Waits for all the specified events.
- eventmask_t chEvtWaitOneTimeout (eventmask_t events, sysinterval_t timeout)

  Waits for exactly one of the specified events.
- eventmask_t chEvtWaitAnyTimeout (eventmask_t events, sysinterval_t timeout)

  Waits for any of the specified events.
- eventmask_t chEvtWaitAllTimeout (eventmask_t events, sysinterval_t timeout)

  Waits for all the specified events.
- static void chEvtObjectInit (event_source_t *esp)

  Initializes an Event Source.
- static void chEvtRegisterMask (event_source_t *esp, event_listener_t *elp, eventmask_t events)

  Registers an Event Listener on an Event Source.
- static void chEvtRegister (event_source_t *esp, event_listener_t *elp, eventid_t event)

  Registers an Event Listener on an Event Source.
- static bool chEvtIsListeningI (event_source_t *esp)

  Verifies if there is at least one `event_listener_t` registered.
- static void chEvtBroadcast (event_source_t *esp)

  Signals all the Event Listeners registered on the specified Event Source.
- static void chEvtBroadcastI (event_source_t *esp)

  Signals all the Event Listeners registered on the specified Event Source.
- static eventmask_t chEvtAddEventsI (eventmask_t events)

  *Adds (OR) a set of events to the current thread, this is **much** faster than using `chEvtBroadcast()` or `chEvtSignal()`.*
- static eventmask_t chEvtGetEventsX (void)

  Returns the events mask.

8.38.1 Detailed Description

Events macros and structures.

8.39 chinstances.h File Reference

OS instances macros and structures.

Macros

- #define __instance_get_currthread(oip)

  Current thread pointer get macro.
- #define __instance_set_currthread(oip, tp)

  Current thread pointer set macro.

Functions

- void chInstanceObjectInit (os_instance_t *oip, const os_instance_config_t *oicp)

  Initializes a system instance.

8.39.1 Detailed Description

OS instances macros and structures.

8.40 chlists.h File Reference

Lists and Queues header.

Data Structures

- struct ch_list

  Structure representing a generic single link list header and element.
- struct ch_queue

  Structure representing a generic bidirectional linked list header and element.
- struct ch_priority_queue

  Structure representing a generic priority-ordered bidirectional linked list header and element.
- struct ch_delta_list

  Delta list element and header structure.

Macros

- #define __CH_QUEUE_DATA(name)

  Data part of a static queue object initializer.
- #define CH_QUEUE_DECL(name)

  Static queue object initializer.

Typedefs

- typedef struct ch_list ch_list_t

  Type of a generic single link list header and element.
- typedef struct ch_queue ch_queue_t

  Type of a generic bidirectional linked list header and element.
- typedef struct ch_priority_queue ch_priority_queue_t

  Type of a generic priority-ordered bidirectional linked list header and element.
- typedef struct ch_delta_list ch_delta_list_t

  Type of a generic bidirectional linked delta list header and element.

Functions

- static void ch_list_init (ch_list_t *lp)

  List initialization.
- static bool ch_list_isempty (ch_list_t *lp)

  Evaluates to `true` if the specified list is empty.
- static bool ch_list_notempty (ch_list_t *lp)

  Evaluates to `true` if the specified list is not empty.
- static void ch_list_link (ch_list_t *lp, ch_list_t *p)

  Pushes an element on top of a stack list.
- static ch_list_t * ch_list_unlink (ch_list_t *lp)

  Pops an element from the top of a stack list and returns it.
- static void ch_queue_init (ch_queue_t *qp)

  Queue initialization.
- static bool ch_queue_isempty (const ch_queue_t *qp)

  Evaluates to `true` if the specified queue is empty.
- static bool ch_queue_notempty (const ch_queue_t *qp)

  Evaluates to `true` if the specified queue is not empty.
- static void ch_queue_insert (ch_queue_t *qp, ch_queue_t *p)

  Inserts an element into a queue.
- static ch_queue_t * ch_queue_fifo_remove (ch_queue_t *qp)

  Removes the first-out element from a queue and returns it.
- static ch_queue_t * ch_queue_lifo_remove (ch_queue_t *qp)

  Removes the last-out element from a queue and returns it.
- static ch_queue_t * ch_queue_dequeue (ch_queue_t *p)

  Removes an element from a queue and returns it.
- static void ch_pqueue_init (ch_priority_queue_t *pqp)

  Priority queue initialization.
- static ch_priority_queue_t * ch_pqueue_remove_highest (ch_priority_queue_t *pqp)

  Removes the highest priority element from a priority queue and returns it.
- static ch_priority_queue_t * ch_pqueue_insert_behind (ch_priority_queue_t *pqp, ch_priority_queue_t *p)

  Inserts an element in the priority queue placing it behind its peers.
- static ch_priority_queue_t * ch_pqueue_insert_ahead (ch_priority_queue_t *pqp, ch_priority_queue_t *p)

  Inserts an element in the priority queue placing it ahead of its peers.
- static void ch_dlist_init (ch_delta_list_t *dlhp)

  Delta list initialization.
- static bool ch_dlist_isempty (ch_delta_list_t *dlhp)

  Evaluates to `true` if the specified delta list is empty.
- static bool ch_dlist_notempty (ch_delta_list_t *dlhp)

  Evaluates to `true` if the specified queue is not empty.
- static bool ch_dlist_islast (ch_delta_list_t *dlhp, ch_delta_list_t *dlp)

  Last element in the delta list check.
- static bool ch_dlist_isfirst (ch_delta_list_t *dlhp, ch_delta_list_t *dlp)

  Fist element in the delta list check.
- static void ch_dlist_insert_after (ch_delta_list_t *dlhp, ch_delta_list_t *dlp, sysinterval_t delta)

  Inserts an element after another header element.
- static void ch_dlist_insert_before (ch_delta_list_t *dlhp, ch_delta_list_t *dlp, sysinterval_t delta)

  Inserts an element before another header element.
- static void ch_dlist_insert (ch_delta_list_t *dlhp, ch_delta_list_t *dlep, sysinterval_t delta)

  Inserts an element in a delta list.
- static ch_delta_list_t * ch_dlist_remove_first (ch_delta_list_t *dlhp)

  Dequeues an element from the delta list.
- static ch_delta_list_t * ch_dlist_dequeue (ch_delta_list_t *dlp)

  Dequeues an element from the delta list.

8.40.1 Detailed Description

Lists and Queues header.

8.41 chmsg.h File Reference

Messages macros and structures.

Macros

- #define __ch_msg_insert(qp, tp)

Functions

- msg_t chMsgSend (thread_t *tp, msg_t msg)

  Sends a message to the specified thread.
- thread_t * chMsgWaitS (void)

  Suspends the thread and waits for an incoming message.
- thread_t * chMsgWaitTimeoutS (sysinterval_t timeout)

  Suspends the thread and waits for an incoming message or a timeout to occur.
- thread_t * chMsgPollS (void)

  Poll to check for an incoming message.
- void chMsgRelease (thread_t *tp, msg_t msg)

  Releases a sender thread specifying a response message.
- static thread_t * chMsgWait (void)

  Suspends the thread and waits for an incoming message.
- static thread_t * chMsgWaitTimeout (sysinterval_t timeout)

  Suspends the thread and waits for an incoming message or a timeout to occur.
- static thread_t * chMsgPoll (void)

  Poll to check for an incoming message.
- static bool chMsgIsPendingI (thread_t *tp)

  Evaluates to `true` *if the thread has pending messages.*
- static msg_t chMsgGet (thread_t *tp)

  Returns the message carried by the specified thread.
- static void chMsgReleaseS (thread_t *tp, msg_t msg)

  Releases the thread waiting on top of the messages queue.

8.41.1 Detailed Description

Messages macros and structures.

8.42 chmtx.h File Reference

Mutexes macros and structures.

Data Structures

- struct ch_mutex

  Mutex structure.

Macros

- #define __MUTEX_DATA(name)

  Data part of a static mutex initializer.
- #define MUTEX_DECL(name)

  Static mutex initializer.

Typedefs

- typedef struct ch_mutex mutex_t

  Type of a mutex structure.

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 chMtxUnlockAll (void)

  Unlocks all mutexes owned by the invoking thread.
- void chMtxUnlockAllS (void)

  Unlocks all mutexes owned by the invoking thread.
- static bool chMtxQueueNotEmptyS (mutex_t *mp)

  Returns `true` if the mutex queue contains at least a waiting thread.
- static thread_t * chMtxGetOwnerI (mutex_t *mp)

  Returns the mutex owner thread.
- static mutex_t * chMtxGetNextMutexX (void)

  Returns the next mutex in the mutexes stack of the current thread.

8.42.1 Detailed Description

Mutexes macros and structures.

8.43 chobjects.h File Reference

Operating System Objects macros and structures.

Data Structures

- struct ch_virtual_timer
  Structure representing a Virtual Timer.
- struct ch_virtual_timers_list
  Type of virtual timers list header.
- struct ch_registry
  Type of a registry structure.
- struct ch_threads_queue
  Type of a threads queue.
- struct ch_thread
  Structure representing a thread.
- struct ch_ready_list
  Type of a ready list header.
- struct ch_os_instance_config
  Type of an system instance configuration.
- struct ch_os_instance
  System instance data structure.
- struct ch_system
  Type of system data structure.

Typedefs

- typedef struct ch_virtual_timer virtual_timer_t
  Type of a Virtual Timer.
- typedef void(∗ vtfunc_t) (virtual_timer_t ∗vtp, void ∗p)
  Type of a Virtual Timer callback function.
- typedef struct ch_virtual_timers_list virtual_timers_list_t
  Type of virtual timers list header.
- typedef struct ch_registry registry_t
  Type of a registry structure.
- typedef thread_t ∗ thread_reference_t
  Type of a thread reference.
- typedef struct ch_threads_queue threads_queue_t
  Type of a threads queue.
- typedef struct ch_ready_list ready_list_t
  Type of a ready list header.
- typedef struct ch_os_instance_config os_instance_config_t
  Type of an system instance configuration.
- typedef struct ch_system ch_system_t
  Type of system data structure.

Enumerations

- enum system_state_t { ch_sys_uninit = 0 , ch_sys_initializing = 1 , ch_sys_running = 2 , ch_sys_halted = 3 }
  Global state of the operating system.

8.43.1 Detailed Description

Operating System Objects macros and structures.

8.44 chport.h File Reference

Port wrapper header.

```
#include "chcore.h"
```

Macros

- #define PORT_CORES_NUMBER 1
- #define CH_PORT_SUPPORTS_RECURSIVE_LOCKS FALSE

8.44.1 Detailed Description

Port wrapper header.

8.45 chregistry.h File Reference

Threads registry macros and structures.

Data Structures

- struct chdebug_t

  ChibiOS/RT memory signature record.

Macros

- #define REG_HEADER(oip)

  Access to the registry list header.
- #define REG_REMOVE(tp)

  Removes a thread from the registry list.
- #define REG_INSERT(oip, tp)

  Adds a thread to the registry list.

Functions

- thread_t * chRegFirstThread (void)
  Returns the first thread in the system.
- thread_t * chRegNextThread (thread_t *tp)
  Returns the thread next to the specified one.
- thread_t * chRegFindThreadByName (const char *name)
  Retrieves a thread pointer by name.
- thread_t * chRegFindThreadByPointer (thread_t *tp)
  Confirms that a pointer is a valid thread pointer.
- thread_t * chRegFindThreadByWorkingArea (stkalign_t *wa)
  Confirms that a working area is being used by some active thread.
- static void __reg_object_init (registry_t *rp)
  Initializes a registry.
- static void chRegSetThreadName (const char *name)
  Sets the current thread name.
- static const char * chRegGetThreadNameX (thread_t *tp)
  Returns the name of the specified thread.
- static void chRegSetThreadNameX (thread_t *tp, const char *name)
  Changes the name of the specified thread.

Variables

- ROMCONST chdebug_t ch_debug

8.45.1 Detailed Description

Threads registry macros and structures.

8.46 chrestrictions.h File Reference

Licensing restrictions header.

Macros

- #define CH_DBG_STATISTICS FALSE
- #define CH_DBG_TRACE_MASK CH_DBG_TRACE_MASK_DISABLED
- #define CH_CFG_ST_TIMEDELTA 0
- #define CH_CFG_USE_TM FALSE
- #define CH_CFG_USE_MUTEXES FALSE
- #define CH_CFG_USE_CONDVARS FALSE
- #define CH_CFG_USE_DYNAMIC FALSE

8.46.1 Detailed Description

Licensing restrictions header.

8.47 chrfcu.h File Reference

Runtime Faults Collection Unit macros and structures.

Data Structures

- struct ch_rfcu

  Type of an RFCU structure.

Macros

- #define CH_RFCU_ALL_FAULTS ((rfcu_mask_t)-1)

  Mask of all faults.

Predefined Faults

- #define CH_RFCU_VT_INSUFFICIENT_DELTA 1U
- #define CH_RFCU_VT_SKIPPED_DEADLINE 2U

Typedefs

- typedef uint32_t rfcu_mask_t

  Type of a faults mask.
- typedef struct ch_rfcu rfcu_t

  Type of an RFCU structure.

Functions

- void chRFCUCollectFaultsI (rfcu_mask_t mask)

  Adds fault flags to the current mask.
- rfcu_mask_t chRFCUGetAndClearFaultsI (rfcu_mask_t mask)

  Returns the current faults mask clearing it.
- static void __rfcu_object_init (rfcu_t *rfcup)

  Runtime Faults Collection Unit initialization.

8.47.1 Detailed Description

Runtime Faults Collection Unit macros and structures.

8.48 chschd.h File Reference

Scheduler macros and structures.

Macros

- #define firstprio(rlp)

  Returns the priority of the first thread on the given ready list.
- #define __sch_get_currthread()

  Current thread pointer get macro.

Wakeup status codes

- #define MSG_OK (msg_t)0

  Normal wakeup message.
- #define MSG_TIMEOUT (msg_t)-1

  Wakeup caused by a timeout condition.
- #define MSG_RESET (msg_t)-2

  Wakeup caused by a reset condition.

Priority constants

- #define NOPRIO (tprio_t)0

  Ready list header priority.
- #define IDLEPRIO (tprio_t)1

  Idle priority.
- #define LOWPRIO (tprio_t)2

  Lowest priority.
- #define NORMALPRIO (tprio_t)128

  Normal priority.
- #define HIGHPRIO (tprio_t)255

  Highest priority.

Thread states

- #define CH_STATE_READY (tstate_t)0

  Waiting on the ready list.
- #define CH_STATE_CURRENT (tstate_t)1

  Currently running.
- #define CH_STATE_WTSTART (tstate_t)2

  Just created.
- #define CH_STATE_SUSPENDED (tstate_t)3

  Suspended state.
- #define CH_STATE_QUEUED (tstate_t)4

  On a queue.
- #define CH_STATE_WTSEM (tstate_t)5

  On a semaphore.
- #define CH_STATE_WTMTX (tstate_t)6

  On a mutex.
- #define CH_STATE_WTCOND (tstate_t)7

  On a cond.variable.
- #define CH_STATE_SLEEPING (tstate_t)8

  Sleeping.
- #define CH_STATE_WTEXIT (tstate_t)9

  Waiting a thread.
- #define CH_STATE_WTOREVT (tstate_t)10

  One event.
- #define CH_STATE_WTANDEVT (tstate_t)11

  Several events.
- #define CH_STATE_SNDMSGQ (tstate_t)12

  Sending a message, in queue.

- #define CH_STATE_SNDMSG (tstate_t)13
  Sent a message, waiting answer.
- #define CH_STATE_WTMSG (tstate_t)14
  Waiting for a message.
- #define CH_STATE_FINAL (tstate_t)15
  Thread terminated.
- #define CH_STATE_NAMES
  Thread states as array of strings.

Thread flags and attributes

- #define CH_FLAG_MODE_MASK (tmode_t)3U
  Thread memory mode mask.
- #define CH_FLAG_MODE_STATIC (tmode_t)0U
  Static thread.
- #define CH_FLAG_MODE_HEAP (tmode_t)1U
  Thread allocated from a Memory Heap.
- #define CH_FLAG_MODE_MPOOL (tmode_t)2U
  Thread allocated from a Memory Pool.
- #define CH_FLAG_TERMINATE (tmode_t)4U
  Termination requested flag.

Functions

- void chSchObjectInit (os_instance_t *oip, const os_instance_config_t *oicp)
- thread_t * chSchReadyI (thread_t *tp)
  Inserts a thread in the Ready List placing it behind its peers.
- void chSchGoSleepS (tstate_t newstate)
  Puts the current thread to sleep into the specified state.
- msg_t chSchGoSleepTimeoutS (tstate_t newstate, sysinterval_t timeout)
  Puts the current thread to sleep into the specified state with timeout specification.
- void chSchWakeupS (thread_t *ntp, msg_t msg)
  Wakes up a thread.
- void chSchRescheduleS (void)
  Performs a reschedule if a higher priority thread is runnable.
- bool chSchIsPreemptionRequired (void)
  Evaluates if preemption is required.
- void chSchDoPreemption (void)
  Switches to the first thread on the runnable queue.
- void chSchPreemption (void)
  All-in-one preemption code.
- void chSchDoYieldS (void)
  Yields the time slot.
- thread_t * chSchSelectFirst (void)
  Makes runnable the fist thread in the ready list, does not reschedule internally.
- void ch_sch_prio_insert (ch_queue_t *qp, ch_queue_t *tp)
  Inserts a thread into a priority ordered queue.

8.48.1 Detailed Description

Scheduler macros and structures.

8.49 chsem.h File Reference

Semaphores macros and structures.

Data Structures

- struct ch_semaphore
  Semaphore structure.

Macros

- #define __SEMAPHORE_DATA(name, n)
  Data part of a static semaphore initializer.
- #define SEMAPHORE_DECL(name, n)
  Static semaphore initializer.

Typedefs

- typedef struct ch_semaphore semaphore_t
  Semaphore structure.

Functions

- void chSemObjectInit (semaphore_t *sp, cnt_t n)
  Initializes a semaphore with the specified counter value.
- void chSemResetWithMessage (semaphore_t *sp, cnt_t n, msg_t msg)
  Performs a reset operation on the semaphore.
- void chSemResetWithMessageI (semaphore_t *sp, cnt_t n, msg_t msg)
  Performs a reset operation on the semaphore.
- msg_t chSemWait (semaphore_t *sp)
  Performs a wait operation on a semaphore.
- msg_t chSemWaitS (semaphore_t *sp)
  Performs a wait operation on a semaphore.
- msg_t chSemWaitTimeout (semaphore_t *sp, sysinterval_t timeout)
  Performs a wait operation on a semaphore with timeout specification.
- msg_t chSemWaitTimeoutS (semaphore_t *sp, sysinterval_t timeout)
  Performs a wait operation on a semaphore with timeout specification.
- void chSemSignal (semaphore_t *sp)
  Performs a signal operation on a semaphore.
- void chSemSignalI (semaphore_t *sp)
  Performs a signal operation on a semaphore.
- void chSemAddCounterI (semaphore_t *sp, cnt_t n)
  Adds the specified value to the semaphore counter.
- msg_t chSemSignalWait (semaphore_t *sps, semaphore_t *spw)
  Performs atomic signal and wait operations on two semaphores.
- static void chSemReset (semaphore_t *sp, cnt_t n)
  Performs a reset operation on the semaphore.
- static void chSemResetI (semaphore_t *sp, cnt_t n)
  Performs a reset operation on the semaphore.
- static void chSemFastWaitI (semaphore_t *sp)
  Decreases the semaphore counter.
- static void chSemFastSignalI (semaphore_t *sp)
  Increases the semaphore counter.
- static cnt_t chSemGetCounterI (const semaphore_t *sp)
  Returns the semaphore counter current value.

8.49.1 Detailed Description

Semaphores macros and structures.

8.50 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)
- void __stats_increase_irq (void)

  Increases the IRQ counter.
- void __stats_ctxswc (thread_t *ntp, thread_t *otp)

  Updates context switch related statistics.
- void __stats_start_measure_crit_thd (void)

  Starts the measurement of a thread critical zone.
- void __stats_stop_measure_crit_thd (void)

  Stops the measurement of a thread critical zone.
- void __stats_start_measure_crit_isr (void)

  Starts the measurement of an ISR critical zone.
- void __stats_stop_measure_crit_isr (void)

  Stops the measurement of an ISR critical zone.
- static void __stats_object_init (kernel_stats_t *ksp)

  Statistics initialization.

8.50.1 Detailed Description

Statistics module macros and structures.

8.51 chsys.h File Reference

System related macros and structures.

Macros

- #define CH_SYS_CORE0_MEMORY PORT_CORE0_BSS_SECTION
  Core zero memory affinity macro.
- #define CH_SYS_CORE1_MEMORY PORT_CORE1_BSS_SECTION
  Core one memory affinity macro.
- #define currcore ch_system.instances[port_get_core_id()]
  Access to current core's instance structure.
- #define chSysGetRealtimeCounterX()
  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)
  Priority level validation macro.
- #define CH_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)
  Standard normal IRQ handler declaration.

Fast ISRs abstraction macros

- #define CH_FAST_IRQ_HANDLER(id)
  Standard fast IRQ handler declaration.

Time conversion utilities for the realtime counter

- #define S2RTC(freq, sec)
  Seconds to realtime counter.
- #define MS2RTC(freq, msec)
  Milliseconds to realtime counter.
- #define US2RTC(freq, usec)
  Microseconds to realtime counter.
- #define RTC2S(freq, n)
  Realtime counter cycles to seconds.
- #define RTC2MS(freq, n)
  Realtime counter cycles to milliseconds.
- #define RTC2US(freq, n)
  Realtime counter cycles to microseconds.

Functions

- void chSysWaitSystemState (system_state_t state)

  Waits for the system state to be equal to the specified one.
- void chSysInit (void)

  System initialization.
- thread_t * chSysGetIdleThreadX (void)

  Returns a pointer to the idle thread.
- bool chSysIntegrityCheckI (unsigned testmask)

  System integrity check.
- void chSysTimerHandlerI (void)

  Handles time ticks for round robin preemption and timer increments.
- syssts_t chSysGetStatusAndLockX (void)

  Returns the execution status and enters a critical zone.
- void chSysRestoreStatusX (syssts_t sts)

  Restores the specified execution status and leaves a critical zone.
- bool chSysIsCounterWithinX (rtcnt_t cnt, rtcnt_t start, rtcnt_t end)

  Realtime window test.
- void chSysPolledDelayX (rtcnt_t cycles)

  Polled delay.
- static void chSysDisable (void)

  Raises the system interrupt priority mask to the maximum level.
- static void chSysSuspend (void)

  Raises the system interrupt priority mask to system level.
- static void chSysEnable (void)

  Lowers the system interrupt priority mask to user level.
- static void chSysLock (void)

  Enters the kernel lock state.
- static void chSysUnlock (void)

  Leaves the kernel lock state.
- static void chSysLockFromISR (void)

  Enters the kernel lock state from within an interrupt handler.
- static void chSysUnlockFromISR (void)

  Leaves the kernel lock state from within an interrupt handler.
- static void chSysUnconditionalLock (void)

  Unconditionally enters the kernel lock state.
- static void chSysUnconditionalUnlock (void)

  Unconditionally leaves the kernel lock state.
- static void chSysNotifyInstance (os_instance_t *oip)

  Notifies an OS instance to check for reschedule.

8.51.1 Detailed Description

System related macros and structures.

8.52 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)

  Data part of a static threads queue object initializer.
- #define THREADS_QUEUE_DECL(name)

  Static threads queue object initializer.

Working Areas

- #define THD_WORKING_AREA_SIZE(n)

  Calculates the total Working Area size.
- #define THD_WORKING_AREA(s, n)

  Static working area allocation.
- #define THD_WORKING_AREA_BASE(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)

  Thread declaration macro.

Threads initializers

- #define THD_DESCRIPTOR(name, wbase, wend, prio, funcp, arg)

  Thread descriptor initializer with no affinity.
- #define THD_DESCRIPTOR_AFFINITY(name, wbase, wend, prio, funcp, arg, oip)

  Thread descriptor initializer with no affinity.

Macro Functions

- #define chThdSleepSeconds(sec)

  Delays the invoking thread for the specified number of seconds.
- #define chThdSleepMilliseconds(msec)

  Delays the invoking thread for the specified number of milliseconds.
- #define chThdSleepMicroseconds(usec)

  Delays the invoking thread for the specified number of microseconds.

Typedefs

- typedef void(∗ tfunc_t) (void ∗p)

  Thread function.

Functions

- thread_t * __thd_object_init (os_instance_t *oip, thread_t *tp, const char *name, tprio_t prio)

  Initializes a thread structure.
- void __thd_stackfill (uint8_t *startp, uint8_t *endp)

  Stack fill utility.
- thread_t * chThdCreateSuspendedI (const thread_descriptor_t *tdp)

  Creates a new thread.
- thread_t * chThdCreateSuspended (const thread_descriptor_t *tdp)

  Creates a new thread.
- thread_t * chThdCreateI (const thread_descriptor_t *tdp)

  Creates a new thread.
- thread_t * chThdCreate (const thread_descriptor_t *tdp)

  Creates a new thread.
- thread_t * chThdCreateStatic (void *wsp, size_t size, tprio_t prio, tfunc_t pf, void *arg)

  Creates a new thread.
- thread_t * chThdStart (thread_t *tp)

  Starts a thread created with `chThdCreateSuspended()`.
- 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_t`.
- static tprio_t chThdGetPriorityX (void)

  Returns the current thread priority.
- static systime_t chThdGetTicksX (thread_t *tp)

  Returns the number of ticks consumed by the specified thread.
- static stkalign_t * chThdGetWorkingAreaX (thread_t *tp)

  Returns the working area base of the specified thread.
- static bool chThdTerminatedX (thread_t *tp)

  Verifies if the specified thread is in the `CH_STATE_FINAL` state.
- static bool chThdShouldTerminateX (void)

  Verifies if the current thread has a termination request pending.
- static thread_t * chThdStartI (thread_t *tp)

  Resumes a thread created with `chThdCreateI()`.
- static void chThdSleepS (sysinterval_t ticks)

  Suspends the invoking thread for the specified number of ticks.
- static void chThdQueueObjectInit (threads_queue_t *tqp)

  Initializes a threads queue object.
- static bool chThdQueueIsEmptyI (threads_queue_t *tqp)

  Evaluates to `true` if the specified queue is empty.
- static void chThdDoDequeueNextI (threads_queue_t *tqp, msg_t msg)

  Dequeues and wakes up one thread from the threads queue object.

8.52.1 Detailed Description

Threads module macros and structures.

8.53 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)
  Seconds to time interval.
- #define TIME_MS2I(msecs)
  Milliseconds to time interval.
- #define TIME_US2I(usecs)
  Microseconds to time interval.
- #define TIME_I2S(interval)
  Time interval to seconds.
- #define TIME_I2MS(interval)
  Time interval to milliseconds.
- #define TIME_I2US(interval)
  Time interval to microseconds.

Typedefs

- typedef uint64_t systime_t
  Type of system time.
- typedef uint64_t sysinterval_t
  Type of time interval.
- typedef uint64_t systimestamp_t
  Type of a time stamp.
- typedef uint32_t time_secs_t
  Type of seconds.
- typedef uint32_t time_msecs_t
  Type of milliseconds.
- typedef uint32_t time_usecs_t
  Type of microseconds.
- typedef uint64_t time_conv_t
  Type of time conversion variable.

Functions

Secure time conversion utilities

- static sysinterval_t chTimeS2I (time_secs_t secs)
  Seconds to time interval.
- static sysinterval_t chTimeMS2I (time_msecs_t msec)
  Milliseconds to time interval.
- static sysinterval_t chTimeUS2I (time_usecs_t usec)
  Microseconds to time interval.
- static time_secs_t chTimeI2S (sysinterval_t interval)
  Time interval to seconds.
- static time_msecs_t chTimeI2MS (sysinterval_t interval)
  Time interval to milliseconds.
- static time_usecs_t chTimeI2US (sysinterval_t interval)
  Time interval to microseconds.
- static systime_t chTimeAddX (systime_t systime, sysinterval_t interval)
  Adds an interval to a system time returning a system time.
- static sysinterval_t chTimeDiffX (systime_t start, systime_t end)
  Subtracts two system times returning an interval.
- static bool chTimeIsInRangeX (systime_t time, systime_t start, systime_t end)
  Checks if the specified time is within the specified time range.
- static systimestamp_t chTimeStampAddX (systimestamp_t stamp, sysinterval_t interval)
  Adds an interval to a time stamp returning a time stamp.
- static sysinterval_t chTimeStampDiffX (systimestamp_t start, systimestamp_t end)
  Subtracts two time stamps returning an interval.
- static bool chTimeStampIsInRangeX (systimestamp_t stamp, systimestamp_t start, systimestamp_t end)
  Checks if the specified time stamp is within the specified time stamps range.

8.53.1 Detailed Description

Time and intervals macros and structures.

8.54 chtm.h File Reference

Time Measurement module macros and structures.

Data Structures

- struct tm_calibration_t

  Type of a time measurement calibration data.
- struct time_measurement_t

  Type of a Time Measurement object.

Macros

- #define TM_CALIBRATION_LOOP 4U

  Number of iterations in the calibration loop.

Functions

- void chTMObjectInit (time_measurement_t *tmp)

  Initializes a `TimeMeasurement` object.
- NOINLINE void chTMStartMeasurementX (time_measurement_t *tmp)

  Starts a measurement.
- NOINLINE void chTMStopMeasurementX (time_measurement_t *tmp)

  Stops a measurement.
- NOINLINE void chTMChainMeasurementToX (time_measurement_t *tmp1, time_measurement_t *tmp2)

  Stops a measurement and chains to the next one using the same time stamp.
- static void __tm_calibration_object_init (tm_calibration_t *tcp)

  Time measurement initialization.

8.54.1 Detailed Description

Time Measurement module macros and structures.

8.55 chtrace.h File Reference

Tracer macros and structures.

Data Structures

- struct trace_event_t

  Trace buffer record.
- struct trace_buffer_t

  Trace buffer header.

Macros

- #define __trace_ready(tp, msg)
- #define __trace_switch(ntp, otp)
- #define __trace_isr_enter(isr)
- #define __trace_isr_leave(isr)
- #define __trace_halt(reason)
- #define chDbgWriteTraceI(up1, up2)
- #define chDbgWriteTrace(up1, up2)

Trace record types

- #define CH_TRACE_TYPE_UNUSED 0U
- #define CH_TRACE_TYPE_READY 1U
- #define CH_TRACE_TYPE_SWITCH 2U
- #define CH_TRACE_TYPE_ISR_ENTER 3U
- #define CH_TRACE_TYPE_ISR_LEAVE 4U
- #define CH_TRACE_TYPE_HALT 5U
- #define CH_TRACE_TYPE_USER 6U

Events to trace

- #define CH_DBG_TRACE_MASK_DISABLED 255U
- #define CH_DBG_TRACE_MASK_NONE 0U
- #define CH_DBG_TRACE_MASK_READY 1U
- #define CH_DBG_TRACE_MASK_SWITCH 2U
- #define CH_DBG_TRACE_MASK_ISR 4U
- #define CH_DBG_TRACE_MASK_HALT 8U
- #define CH_DBG_TRACE_MASK_USER 16U
- #define CH_DBG_TRACE_MASK_SLOW
- #define CH_DBG_TRACE_MASK_ALL

Debug related settings

- #define CH_DBG_TRACE_MASK CH_DBG_TRACE_MASK_DISABLED

  Trace buffer entries.
- #define CH_DBG_TRACE_BUFFER_SIZE 128

  Trace buffer entries.

Functions

- void __trace_object_init (trace_buffer_t *tbp)

  Circular trace buffer initialization.
- void __trace_ready (thread_t *tp, msg_t msg)

  Inserts in the circular debug trace buffer a ready record.
- void __trace_switch (thread_t *ntp, thread_t *otp)

  Inserts in the circular debug trace buffer a context switch record.
- void __trace_isr_enter (const char *isr)

  Inserts in the circular debug trace buffer an ISR-enter record.
- void __trace_isr_leave (const char *isr)

  Inserts in the circular debug trace buffer an ISR-leave record.
- void __trace_halt (const char *reason)

  Inserts in the circular debug trace buffer an halt record.
- void chTraceWriteI (void *up1, void *up2)

  Adds an user trace record to the trace buffer.
- void chTraceWrite (void *up1, void *up2)

  Adds an user trace record to the trace buffer.
- void chTraceSuspendI (uint16_t mask)

  Suspends one or more trace events.
- void chTraceSuspend (uint16_t mask)

  Suspends one or more trace events.
- void chTraceResumeI (uint16_t mask)

  Resumes one or more trace events.
- void chTraceResume (uint16_t mask)

  Resumes one or more trace events.

8.55.1 Detailed Description

Tracer macros and structures.

8.56 chvt.h File Reference

Time and Virtual Timers module macros and structures.

Functions

- void chVTDoSetI (virtual_timer_t *vtp, sysinterval_t delay, vtfunc_t vtfunc, void *par)

  Enables a one-shot virtual timer.
- void chVTDoSetContinuousI (virtual_timer_t *vtp, sysinterval_t delay, vtfunc_t vtfunc, void *par)

  Enables a continuous virtual timer.
- void chVTDoResetI (virtual_timer_t *vtp)

  Disables a Virtual Timer.
- sysinterval_t chVTGetRemainingIntervalI (virtual_timer_t *vtp)

  Returns the remaining time interval before next timer trigger.
- void chVTDoTickI (void)

  Virtual timers ticker.

- systimestamp_t chVTGetTimeStampI (void)

  Generates a monotonic time stamp.
- void chVTResetTimeStampI (void)

  Resets and re-synchronizes the time stamps monotonic counter.
- static void chVTObjectInit (virtual_timer_t *vtp)

  Initializes a `virtual_timer_t` object.
- static systime_t chVTGetSystemTimeX (void)

  Current system time.
- static systime_t chVTGetSystemTime (void)

  Current system time.
- static sysinterval_t chVTTimeElapsedSinceX (systime_t start)

  Returns the elapsed time since the specified start time.
- static bool chVTIsSystemTimeWithinX (systime_t start, systime_t end)

  Checks if the current system time is within the specified time window.
- static bool chVTIsSystemTimeWithin (systime_t start, systime_t end)

  Checks if the current system time is within the specified time window.
- static bool chVTGetTimersStateI (sysinterval_t *timep)

  Returns the time interval until the next timer event.
- static bool chVTIsArmedI (const virtual_timer_t *vtp)

  Returns `true` if the specified timer is armed.
- static bool chVTIsArmed (const virtual_timer_t *vtp)

  Returns `true` if the specified timer is armed.
- static void chVTResetI (virtual_timer_t *vtp)

  Disables a Virtual Timer.
- static void chVTReset (virtual_timer_t *vtp)

  Disables a Virtual Timer.
- static void chVTSetI (virtual_timer_t *vtp, sysinterval_t delay, vtfunc_t vtfunc, void *par)

  Enables a one-shot virtual timer.
- static void chVTSet (virtual_timer_t *vtp, sysinterval_t delay, vtfunc_t vtfunc, void *par)

  Enables a one-shot virtual timer.
- static void chVTSetContinuousI (virtual_timer_t *vtp, sysinterval_t delay, vtfunc_t vtfunc, void *par)

  Enables a continuous virtual timer.
- static void chVTSetContinuous (virtual_timer_t *vtp, sysinterval_t delay, vtfunc_t vtfunc, void *par)

  Enables a continuous virtual timer.
- static sysinterval_t chVTGetReloadIntervalX (virtual_timer_t *vtp)

  Returns the current reload value.
- static void chVTSetReloadIntervalX (virtual_timer_t *vtp, sysinterval_t reload)

  Changes a timer reload time interval.
- static systimestamp_t chVTGetTimeStamp (void)

  Generates a monotonic time stamp.
- static void chVTResetTimeStamp (void)

  Resets and re-synchronizes the time stamps monotonic counter.
- static void __vt_object_init (virtual_timers_list_t *vtlp)

  Virtual Timers instance initialization.

8.56.1 Detailed Description

Time and Virtual Timers module macros and structures.

8.57 chcond.c File Reference

Condition Variables code.

```
#include "ch.h"
```

Functions

- void chCondObjectInit (condition_variable_t *cp)

  Initializes s condition_variable_t structure.
- void chCondSignal (condition_variable_t *cp)

  Signals one thread that is waiting on the condition variable.
- void chCondSignalI (condition_variable_t *cp)

  Signals one thread that is waiting on the condition variable.
- void chCondBroadcast (condition_variable_t *cp)

  Signals all threads that are waiting on the condition variable.
- void chCondBroadcastI (condition_variable_t *cp)

  Signals all threads that are waiting on the condition variable.
- msg_t chCondWait (condition_variable_t *cp)

  Waits on the condition variable releasing the mutex lock.
- msg_t chCondWaitS (condition_variable_t *cp)

  Waits on the condition variable releasing the mutex lock.
- msg_t chCondWaitTimeout (condition_variable_t *cp, sysinterval_t timeout)

  Waits on the condition variable releasing the mutex lock.
- msg_t chCondWaitTimeoutS (condition_variable_t *cp, sysinterval_t timeout)

  Waits on the condition variable releasing the mutex lock.

8.57.1 Detailed Description

Condition Variables code.

8.58 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.

8.58.1 Detailed Description

Debug support code.

8.59 chdynamic.c File Reference

Dynamic threads code.

```
#include "ch.h"
```

Functions

- thread_t * chThdCreateFromHeap (memory_heap_t *heapp, size_t size, const char *name, tprio_t prio, tfunc_t pf, void *arg)

  Creates a new thread allocating the memory from the heap.
- thread_t * chThdCreateFromMemoryPool (memory_pool_t *mp, const char *name, tprio_t prio, tfunc_t pf, void *arg)

  Creates a new thread allocating the memory from the specified memory pool.

8.59.1 Detailed Description

Dynamic threads code.

8.60 chevents.c File Reference

Events code.

```
#include "ch.h"
```

Functions

- void chEvtRegisterMaskWithFlagsI (event_source_t *esp, event_listener_t *elp, eventmask_t events, eventflags_t wflags)

  Registers an Event Listener on an Event Source.
- void chEvtRegisterMaskWithFlags (event_source_t *esp, event_listener_t *elp, eventmask_t events, eventflags_t wflags)

  Registers an Event Listener on an Event Source.
- void chEvtUnregister (event_source_t *esp, event_listener_t *elp)

  Unregisters an Event Listener from its Event Source.
- eventmask_t chEvtGetAndClearEventsI (eventmask_t events)

  Clears the pending events specified in the events mask.
- eventmask_t chEvtGetAndClearEvents (eventmask_t events)

  Clears the pending events specified in the events mask.
- eventmask_t chEvtAddEvents (eventmask_t events)

  *Adds (OR) a set of events to the current thread, this is **much** faster than using `chEvtBroadcast()` or `chEvtSignal()`.*
- eventflags_t chEvtGetAndClearFlagsI (event_listener_t *elp)

  Returns the unmasked flags associated to an `event_listener_t`.
- eventflags_t chEvtGetAndClearFlags (event_listener_t *elp)

  Returns the flags associated to an `event_listener_t`.
- void chEvtSignalI (thread_t *tp, eventmask_t events)

  Adds a set of event flags directly to the specified `thread_t`.
- void chEvtSignal (thread_t *tp, eventmask_t events)

  Adds a set of event flags directly to the specified `thread_t`.
- void chEvtBroadcastFlagsI (event_source_t *esp, eventflags_t flags)

  Signals all the Event Listeners registered on the specified Event Source.
- void chEvtBroadcastFlags (event_source_t *esp, eventflags_t flags)

  Signals all the Event Listeners registered on the specified Event Source.
- void chEvtDispatch (const evhandler_t *handlers, eventmask_t events)

  Invokes the event handlers associated to an event flags mask.
- eventmask_t chEvtWaitOne (eventmask_t events)

  Waits for exactly one of the specified events.
- eventmask_t chEvtWaitAny (eventmask_t events)

  Waits for any of the specified events.
- eventmask_t chEvtWaitAll (eventmask_t events)

  Waits for all the specified events.
- eventmask_t chEvtWaitOneTimeout (eventmask_t events, sysinterval_t timeout)

  Waits for exactly one of the specified events.
- eventmask_t chEvtWaitAnyTimeout (eventmask_t events, sysinterval_t timeout)

  Waits for any of the specified events.
- eventmask_t chEvtWaitAllTimeout (eventmask_t events, sysinterval_t timeout)

  Waits for all the specified events.

8.60.1 Detailed Description

Events code.

8.61 chinstances.c File Reference

OS instances code.

```
#include "ch.h"
```

Functions

- static void __idle_thread (void *p)
  This function implements the idle thread infinite loop.
- void chInstanceObjectInit (os_instance_t *oip, const os_instance_config_t *oicp)
  Initializes a system instance.

8.61.1 Detailed Description

OS instances code.

8.62 chmsg.c File Reference

Messages code.

```
#include "ch.h"
```

Functions

- msg_t chMsgSend (thread_t *tp, msg_t msg)
  Sends a message to the specified thread.
- thread_t * chMsgWaitS (void)
  Suspends the thread and waits for an incoming message.
- thread_t * chMsgWaitTimeoutS (sysinterval_t timeout)
  Suspends the thread and waits for an incoming message or a timeout to occur.
- thread_t * chMsgPollS (void)
  Poll to check for an incoming message.
- void chMsgRelease (thread_t *tp, msg_t msg)
  Releases a sender thread specifying a response message.

8.62.1 Detailed Description

Messages code.

8.63 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.

8.63.1 Detailed Description

Mutexes code.

8.64 chregistry.c File Reference

Threads registry code.

```
#include <string.h>
#include "ch.h"
```

Functions

- thread_t * chRegFirstThread (void)
  Returns the first thread in the system.
- thread_t * chRegNextThread (thread_t *tp)
  Returns the thread next to the specified one.
- thread_t * chRegFindThreadByName (const char *name)
  Retrieves a thread pointer by name.
- thread_t * chRegFindThreadByPointer (thread_t *tp)
  Confirms that a pointer is a valid thread pointer.
- thread_t * chRegFindThreadByWorkingArea (stkalign_t *wa)
  Confirms that a working area is being used by some active thread.

Variables

- ROMCONST chdebug_t ch_debug

8.64.1 Detailed Description

Threads registry code.

8.65 chrfcu.c File Reference

Runtime Faults Collection Unit code.

```
#include "ch.h"
```

Functions

- void chRFCUCollectFaultsI (rfcu_mask_t mask)
  Adds fault flags to the current mask.
- rfcu_mask_t chRFCUGetAndClearFaultsI (rfcu_mask_t mask)
  Returns the current faults mask clearing it.

8.65.1 Detailed Description

Runtime Faults Collection Unit code.

8.66 chschd.c File Reference

Scheduler code.

```
#include "ch.h"
```

Functions

- static thread_t * __sch_ready_behind (thread_t *tp)

  Inserts a thread in the Ready List placing it behind its peers.
- static thread_t * __sch_ready_ahead (thread_t *tp)

  Inserts a thread in the Ready List placing it ahead its peers.
- static void __sch_reschedule_behind (void)

  Switches to the first thread on the runnable queue.
- static void __sch_reschedule_ahead (void)

  Switches to the first thread on the runnable queue.
- static void __sch_wakeup (virtual_timer_t *vtp, void *p)
- void ch_sch_prio_insert (ch_queue_t *qp, ch_queue_t *tp)

  Inserts a thread into a priority ordered queue.
- thread_t * chSchReadyI (thread_t *tp)

  Inserts a thread in the Ready List placing it behind its peers.
- void chSchGoSleepS (tstate_t newstate)

  Puts the current thread to sleep into the specified state.
- msg_t chSchGoSleepTimeoutS (tstate_t newstate, sysinterval_t timeout)

  Puts the current thread to sleep into the specified state with timeout specification.
- void chSchWakeupS (thread_t *ntp, msg_t msg)

  Wakes up a thread.
- void chSchRescheduleS (void)

  Performs a reschedule if a higher priority thread is runnable.
- bool chSchIsPreemptionRequired (void)

  Evaluates if preemption is required.
- void chSchDoPreemption (void)

  Switches to the first thread on the runnable queue.
- void chSchPreemption (void)

  All-in-one preemption code.
- void chSchDoYieldS (void)

  Yields the time slot.
- thread_t * chSchSelectFirst (void)

  Makes runnable the fist thread in the ready list, does not reschedule internally.

8.66.1 Detailed Description

Scheduler code.

8.67 chsem.c File Reference

Semaphores code.

```
#include "ch.h"
```

Macros

- #define sem_insert(qp, tp)

Functions

- void chSemObjectInit (semaphore_t *sp, cnt_t n)
  Initializes a semaphore with the specified counter value.
- void chSemResetWithMessage (semaphore_t *sp, cnt_t n, msg_t msg)
  Performs a reset operation on the semaphore.
- void chSemResetWithMessageI (semaphore_t *sp, cnt_t n, msg_t msg)
  Performs a reset operation on the semaphore.
- msg_t chSemWait (semaphore_t *sp)
  Performs a wait operation on a semaphore.
- msg_t chSemWaitS (semaphore_t *sp)
  Performs a wait operation on a semaphore.
- msg_t chSemWaitTimeout (semaphore_t *sp, sysinterval_t timeout)
  Performs a wait operation on a semaphore with timeout specification.
- msg_t chSemWaitTimeoutS (semaphore_t *sp, sysinterval_t timeout)
  Performs a wait operation on a semaphore with timeout specification.
- void chSemSignal (semaphore_t *sp)
  Performs a signal operation on a semaphore.
- void chSemSignalI (semaphore_t *sp)
  Performs a signal operation on a semaphore.
- void chSemAddCounterI (semaphore_t *sp, cnt_t n)
  Adds the specified value to the semaphore counter.
- msg_t chSemSignalWait (semaphore_t *sps, semaphore_t *spw)
  Performs atomic signal and wait operations on two semaphores.

8.67.1 Detailed Description

Semaphores code.

8.68 chstats.c File Reference

Statistics module code.

```
#include "ch.h"
```

Functions

- void __stats_increase_irq (void)
  Increases the IRQ counter.
- void __stats_ctxswc (thread_t *ntp, thread_t *otp)
  Updates context switch related statistics.
- void __stats_start_measure_crit_thd (void)
  Starts the measurement of a thread critical zone.
- void __stats_stop_measure_crit_thd (void)
  Stops the measurement of a thread critical zone.
- void __stats_start_measure_crit_isr (void)
  Starts the measurement of an ISR critical zone.
- void __stats_stop_measure_crit_isr (void)
  Stops the measurement of an ISR critical zone.

8.68.1 Detailed Description

Statistics module code.

8.69 chsys.c File Reference

System related code.

```
#include "ch.h"
```

Functions

- static CH_SYS_CORE0_MEMORY THD_WORKING_AREA (ch_c0_idle_thread_wa, PORT_IDLE_↩ THREAD_STACK_SIZE)

  Working area for core 0 idle thread.
- static CH_SYS_CORE1_MEMORY THD_WORKING_AREA (ch_c1_idle_thread_wa, PORT_IDLE_↩ THREAD_STACK_SIZE)

  Working area for core 1 idle thread.
- void chSysWaitSystemState (system_state_t state)

  Waits for the system state to be equal to the specified one.
- void chSysInit (void)

  System initialization.
- void chSysHalt (const char *reason)

  Halts the system.
- thread_t * chSysGetIdleThreadX (void)

  Returns a pointer to the idle thread.
- bool chSysIntegrityCheckI (unsigned testmask)

  System integrity check.
- void chSysTimerHandlerI (void)

  Handles time ticks for round robin preemption and timer increments.
- syssts_t chSysGetStatusAndLockX (void)

  Returns the execution status and enters a critical zone.
- void chSysRestoreStatusX (syssts_t sts)

  Restores the specified execution status and leaves a critical zone.
- bool chSysIsCounterWithinX (rtcnt_t cnt, rtcnt_t start, rtcnt_t end)

  Realtime window test.
- void chSysPolledDelayX (rtcnt_t cycles)

  Polled delay.

Variables

- ch_system_t ch_system

  System root object.
- CH_SYS_CORE0_MEMORY os_instance_t ch0

  Core 0 OS instance.
- stkalign_t __main_thread_stack_base__
- stkalign_t __main_thread_stack_end__
- const os_instance_config_t ch_core0_cfg

  Core 0 OS instance configuration.
- CH_SYS_CORE1_MEMORY os_instance_t ch1

  Core 1 OS instance.
- stkalign_t __c1_main_thread_stack_base__
- stkalign_t __c1_main_thread_stack_end__
- const os_instance_config_t ch_core1_cfg

  Core 1 OS instance configuration.

8.69.1 Detailed Description

System related code.

8.70 chthreads.c File Reference

Threads code.

```
#include "ch.h"
```

Functions

- thread_t * __thd_object_init (os_instance_t *oip, thread_t *tp, const char *name, tprio_t prio)

  Initializes a thread structure.
- void __thd_stackfill (uint8_t *startp, uint8_t *endp)

  Stack fill utility.
- thread_t * chThdCreateSuspendedI (const thread_descriptor_t *tdp)

  Creates a new thread.
- thread_t * chThdCreateSuspended (const thread_descriptor_t *tdp)

  Creates a new thread.
- thread_t * chThdCreateI (const thread_descriptor_t *tdp)

  Creates a new thread.
- thread_t * chThdCreate (const thread_descriptor_t *tdp)

  Creates a new thread.
- thread_t * chThdCreateStatic (void *wsp, size_t size, tprio_t prio, tfunc_t pf, void *arg)

  Creates a new thread.
- thread_t * chThdStart (thread_t *tp)

  Starts a thread created with `chThdCreateSuspended()`.
- 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.

8.70.1 Detailed Description

Threads code.

8.71 chtm.c File Reference

Time Measurement module code.

```
#include "ch.h"
```

Functions

- static void tm_stop (time_measurement_t *tmp, rtcnt_t now, rtcnt_t offset)
- void chTMObjectInit (time_measurement_t *tmp)

  Initializes a `TimeMeasurement` object.
- NOINLINE void chTMStartMeasurementX (time_measurement_t *tmp)

  Starts a measurement.
- NOINLINE void chTMStopMeasurementX (time_measurement_t *tmp)

  Stops a measurement.
- NOINLINE void chTMChainMeasurementToX (time_measurement_t *tmp1, time_measurement_t *tmp2)

  Stops a measurement and chains to the next one using the same time stamp.

8.71.1 Detailed Description

Time Measurement module code.

8.72 chtrace.c File Reference

Tracer code.

```
#include "ch.h"
```

Functions

- static NOINLINE void trace_next (os_instance_t *oip)

  Writes a time stamp and increases the trace buffer pointer.
- void __trace_object_init (trace_buffer_t *tbp)

  Circular trace buffer initialization.
- void __trace_ready (thread_t *tp, msg_t msg)

  Inserts in the circular debug trace buffer a ready record.
- void __trace_switch (thread_t *ntp, thread_t *otp)

  Inserts in the circular debug trace buffer a context switch record.
- void __trace_isr_enter (const char *isr)

  Inserts in the circular debug trace buffer an ISR-enter record.
- void __trace_isr_leave (const char *isr)

  Inserts in the circular debug trace buffer an ISR-leave record.
- void __trace_halt (const char *reason)

  Inserts in the circular debug trace buffer an halt record.
- void chTraceWriteI (void *up1, void *up2)

  Adds an user trace record to the trace buffer.
- void chTraceWrite (void *up1, void *up2)

  Adds an user trace record to the trace buffer.
- void chTraceSuspendI (uint16_t mask)

  Suspends one or more trace events.
- void chTraceSuspend (uint16_t mask)

  Suspends one or more trace events.
- void chTraceResumeI (uint16_t mask)

  Resumes one or more trace events.
- void chTraceResume (uint16_t mask)

  Resumes one or more trace events.

8.72.1 Detailed Description

Tracer code.

8.73 chvt.c File Reference

Time and Virtual Timers module code.

```
#include <string.h>
#include "ch.h"
```

Functions

- static void vt_set_alarm (systime_t now, sysinterval_t delay)
  Alarm time setup.
- static void vt_insert_first (virtual_timers_list_t *vtlp, virtual_timer_t *vtp, systime_t now, sysinterval_t delay)
  Inserts a timer as first element in a delta list.
- static void vt_enqueue (virtual_timers_list_t *vtlp, virtual_timer_t *vtp, sysinterval_t delay)
  Enqueues a virtual timer in a virtual timers list.
- void chVTDoSetI (virtual_timer_t *vtp, sysinterval_t delay, vtfunc_t vtfunc, void *par)
  Enables a one-shot virtual timer.
- void chVTDoSetContinuousI (virtual_timer_t *vtp, sysinterval_t delay, vtfunc_t vtfunc, void *par)
  Enables a continuous virtual timer.
- void chVTDoResetI (virtual_timer_t *vtp)
  Disables a Virtual Timer.
- sysinterval_t chVTGetRemainingIntervalI (virtual_timer_t *vtp)
  Returns the remaining time interval before next timer trigger.
- void chVTDoTickI (void)
  Virtual timers ticker.
- systimestamp_t chVTGetTimeStampI (void)
  Generates a monotonic time stamp.
- void chVTResetTimeStampI (void)
  Resets and re-synchronizes the time stamps monotonic counter.

8.73.1 Detailed Description

Time and Virtual Timers module code.

8.74 chconf.h File Reference

Configuration file template.

Macros

- #define _CHIBIOS_RT_CONF_
- #define _CHIBIOS_RT_CONF_VER_7_0_

System settings

- #define CH_CFG_SMP_MODE FALSE
  Handling of instances.

System timers settings

- #define CH_CFG_ST_RESOLUTION 32
  System time counter resolution.
- #define CH_CFG_ST_FREQUENCY 10000
  System tick frequency.
- #define CH_CFG_INTERVALS_SIZE 32
  Time intervals data size.
- #define CH_CFG_TIME_TYPES_SIZE 32
  Time types data size.
- #define CH_CFG_ST_TIMEDELTA 2
  Time delta constant for the tick-less mode.

Kernel parameters and options

- #define CH_CFG_TIME_QUANTUM 0
  Round robin interval.
- #define CH_CFG_NO_IDLE_THREAD FALSE
  Idle thread automatic spawn suppression.

Performance options

- #define CH_CFG_OPTIMIZE_SPEED TRUE
  OS optimization.

Subsystem options

- #define CH_CFG_USE_TM TRUE
  Time Measurement APIs.
- #define CH_CFG_USE_TIMESTAMP TRUE
  Time Stamps APIs.
- #define CH_CFG_USE_REGISTRY TRUE
  Threads registry APIs.
- #define CH_CFG_USE_WAITEXIT TRUE
  Threads synchronization APIs.
- #define CH_CFG_USE_SEMAPHORES TRUE
  Semaphores APIs.
- #define CH_CFG_USE_SEMAPHORES_PRIORITY FALSE
  Semaphores queuing mode.
- #define CH_CFG_USE_MUTEXES TRUE
  Mutexes APIs.
- #define CH_CFG_USE_MUTEXES_RECURSIVE FALSE
  Enables recursive behavior on mutexes.
- #define CH_CFG_USE_CONDVARS TRUE
  Conditional Variables APIs.
- #define CH_CFG_USE_CONDVARS_TIMEOUT TRUE

Conditional Variables APIs with timeout.

- #define CH_CFG_USE_EVENTS TRUE
  Events Flags APIs.
- #define CH_CFG_USE_EVENTS_TIMEOUT TRUE
  Events Flags APIs with timeout.
- #define CH_CFG_USE_MESSAGES TRUE
  Synchronous Messages APIs.
- #define CH_CFG_USE_MESSAGES_PRIORITY FALSE
  Synchronous Messages queuing mode.
- #define CH_CFG_USE_DYNAMIC TRUE
  Dynamic Threads APIs.

OSLIB options

- #define CH_CFG_USE_MAILBOXES TRUE
  Mailboxes APIs.
- #define CH_CFG_USE_MEMCORE TRUE
  Core Memory Manager APIs.
- #define CH_CFG_MEMCORE_SIZE 0
  Managed RAM size.
- #define CH_CFG_USE_HEAP TRUE
  Heap Allocator APIs.
- #define CH_CFG_USE_MEMPOOLS TRUE
  Memory Pools Allocator APIs.
- #define CH_CFG_USE_OBJ_FIFOS TRUE
  Objects FIFOs APIs.
- #define CH_CFG_USE_PIPES TRUE
  Pipes APIs.
- #define CH_CFG_USE_OBJ_CACHES TRUE
  Objects Caches APIs.
- #define CH_CFG_USE_DELEGATES TRUE
  Delegate threads APIs.
- #define CH_CFG_USE_JOBS TRUE
  Jobs Queues APIs.

Objects factory options

- #define CH_CFG_USE_FACTORY TRUE
  Objects Factory APIs.
- #define CH_CFG_FACTORY_MAX_NAMES_LENGTH 8
  Maximum length for object names.
- #define CH_CFG_FACTORY_OBJECTS_REGISTRY TRUE
  Enables the registry of generic objects.
- #define CH_CFG_FACTORY_GENERIC_BUFFERS TRUE
  Enables factory for generic buffers.
- #define CH_CFG_FACTORY_SEMAPHORES TRUE
  Enables factory for semaphores.
- #define CH_CFG_FACTORY_MAILBOXES TRUE
  Enables factory for mailboxes.
- #define CH_CFG_FACTORY_OBJ_FIFOS TRUE
  Enables factory for objects FIFOs.
- #define CH_CFG_FACTORY_PIPES TRUE
  Enables factory for Pipes.

Debug options

- #define CH_DBG_STATISTICS FALSE

Debug option, kernel statistics.

- #define CH_DBG_SYSTEM_STATE_CHECK TRUE
  Debug option, system state check.
- #define CH_DBG_ENABLE_CHECKS TRUE
  Debug option, parameters checks.
- #define CH_DBG_ENABLE_ASSERTS TRUE
  Debug option, consistency checks.
- #define CH_DBG_TRACE_MASK CH_DBG_TRACE_MASK_ALL
  Debug option, trace buffer.
- #define CH_DBG_TRACE_BUFFER_SIZE 128
  Trace buffer entries.
- #define CH_DBG_ENABLE_STACK_CHECK TRUE
  Debug option, stack checks.
- #define CH_DBG_FILL_THREADS TRUE
  Debug option, stacks initialization.
- #define CH_DBG_THREADS_PROFILING FALSE
  Debug option, threads profiling.

Kernel hooks

- #define CH_CFG_SYSTEM_EXTRA_FIELDS /∗ Add system custom fields here.∗/
  System structure extension.
- #define CH_CFG_SYSTEM_INIT_HOOK()
  System initialization hook.
- #define CH_CFG_OS_INSTANCE_EXTRA_FIELDS /∗ Add OS instance custom fields here.∗/
  OS instance structure extension.
- #define CH_CFG_OS_INSTANCE_INIT_HOOK(oip)
  OS instance initialization hook.
- #define CH_CFG_THREAD_EXTRA_FIELDS /∗ Add threads custom fields here.∗/
  Threads descriptor structure extension.
- #define CH_CFG_THREAD_INIT_HOOK(tp)
  Threads initialization hook.
- #define CH_CFG_THREAD_EXIT_HOOK(tp)
  Threads finalization hook.
- #define CH_CFG_CONTEXT_SWITCH_HOOK(ntp, otp)
  Context switch hook.
- #define CH_CFG_IRQ_PROLOGUE_HOOK()
  ISR enter hook.
- #define CH_CFG_IRQ_EPILOGUE_HOOK()
  ISR exit hook.
- #define CH_CFG_IDLE_ENTER_HOOK()
  Idle thread enter hook.
- #define CH_CFG_IDLE_LEAVE_HOOK()
  Idle thread leave hook.
- #define CH_CFG_IDLE_LOOP_HOOK()
  Idle Loop hook.
- #define CH_CFG_SYSTEM_TICK_HOOK()
  System tick event hook.
- #define CH_CFG_SYSTEM_HALT_HOOK(reason)
  System halt hook.
- #define CH_CFG_TRACE_HOOK(tep)
  Trace hook.
- #define CH_CFG_RUNTIME_FAULTS_HOOK(mask)
  Runtime Faults Collection Unit hook.

8.74.1 Detailed Description

Configuration file template.

A copy of this file must be placed in each project directory, it contains the application specific kernel settings.

Index

_CHIBIOS_RT_CONF_
    Options, 39
_CHIBIOS_RT_CONF_VER_7_0_
    Options, 39
__BSEMAPHORE_DATA
    Binary Semaphores, 319
__CHIBIOS_OSLIB__
    Version Numbers and Identification, 315
__CHIBIOS_RT__
    Version Numbers and Identification, 34
__CHIBIOS__
    Release Information, 25
__CH_OFFSETOF
    OS Types and Structures, 61
__CH_QUEUE_DATA
    Lists and Queues, 74
__CH_STRINGIFY
    OS Types and Structures, 60
__CH_USED
    OS Types and Structures, 61
__CONDVAR_DATA
    Condition Variables, 234
__EVENTSOURCE_DATA
    Event Flags, 246
__GUARDEDMEMORYPOOL_DATA
    Memory Pools, 394
__MAILBOX_DATA
    Mailboxes, 330
__MEMORYPOOL_DATA
    Memory Pools, 394
__MUTEX_DATA
    Mutexes, 223
__PIPE_DATA
    Pipes, 346
__SEMAPHORE_DATA
    Counting Semaphores, 208
__THREADS_QUEUE_DATA
    Threads, 169
__c1_main_thread_stack_base__
    System Management, 125
__c1_main_thread_stack_end__
    System Management, 125
__ch_delegate_fn0
    Delegate Threads, 355
__ch_delegate_fn1
    Delegate Threads, 356
__ch_delegate_fn2
    Delegate Threads, 356
__ch_delegate_fn3
    Delegate Threads, 356
__ch_delegate_fn4
    Delegate Threads, 357
__ch_msg_insert
    Synchronous Messages, 268
__core_init
    Core Memory Manager, 378
__dbg_check_disable
    Checks and Assertions, 291, 294
__dbg_check_enable
    Checks and Assertions, 291, 294
__dbg_check_enter_isr
    Checks and Assertions, 292, 297
__dbg_check_leave_isr
    Checks and Assertions, 292, 297
__dbg_check_lock
    Checks and Assertions, 291, 295
__dbg_check_lock_from_isr
    Checks and Assertions, 292, 296
__dbg_check_suspend
    Checks and Assertions, 291, 294
__dbg_check_unlock
    Checks and Assertions, 291, 295
__dbg_check_unlock_from_isr
    Checks and Assertions, 292, 296
__dbg_object_init
    Checks and Assertions, 299
__factory_init
    Dynamic Objects Factory, 449
__heap_init
    Memory Heaps, 388
__idle_thread
    OS Instances, 68
__instance_get_currthread
    OS Instances, 68
__instance_set_currthread
    OS Instances, 68
__main_thread_stack_base__
    System Management, 125
__main_thread_stack_end__
    System Management, 125
__oslib_init
    Version Numbers and Identification, 317
__reg_object_init
    Registry, 286
__rfcu_object_init
    Runtime Faults Collection Unit, 72
__sch_get_currthread
    Scheduler, 91

__sch_ready_ahead
Scheduler, 92
__sch_ready_behind
Scheduler, 91
__sch_reschedule_ahead
Scheduler, 93
__sch_reschedule_behind
Scheduler, 93
__sch_wakeup
Scheduler, 94
__stats_ctxswc
Statistics, 311
__stats_increase_irq
Statistics, 311
__stats_init
Statistics, 313
__stats_object_init
Statistics, 313
__stats_start_measure_crit_isr
Statistics, 312
__stats_start_measure_crit_thd
Statistics, 311
__stats_stop_measure_crit_isr
Statistics, 312
__stats_stop_measure_crit_thd
Statistics, 312
__thd_object_init
Threads, 174
__thd_stackfill
Threads, 174
__tm_calibration_object_init
Time Measurement, 205
__trace_halt
Tracing, 304, 307
__trace_isr_enter
Tracing, 303, 306
__trace_isr_leave
Tracing, 304, 306
__trace_object_init
Tracing, 304
__trace_ready
Tracing, 303, 305
__trace_switch
Tracing, 303, 305
__vt_object_init
Virtual Timers, 165

align
memory_pool_t, 547
ALL_EVENTS
Event Flags, 246
arg
thread_descriptor_t, 553

Base Kernel Services, 103
basemem
memcore_t, 543
best
time_measurement_t, 555

Binary Semaphores, 318
__BSEMAPHORE_DATA, 319
binary_semaphore_t, 320
BSEMAPHORE_DECL, 320
chBSemGetStateI, 328
chBSemObjectInit, 320
chBSemReset, 325
chBSemResetI, 324
chBSemSignal, 326
chBSemSignalI, 326
chBSemWait, 321
chBSemWaitS, 321
chBSemWaitTimeout, 323
chBSemWaitTimeoutS, 322
binary_semaphore_t
Binary Semaphores, 320
BSEMAPHORE_DECL
Binary Semaphores, 320
buf_list
ch_objects_factory, 490
buffer
mailbox_t, 542
pipe_t, 549
trace_buffer_t, 558

cache_sem
ch_objects_cache, 488
CC_ALIGN_CODE
Compiler portability., 22
CC_ALIGN_DATA
Compiler portability., 22
CC_FORCE_INLINE
Compiler portability., 23
CC_LIKELY
Compiler portability., 24
CC_NO_INLINE
Compiler portability., 23
CC_NO_RETURN
Compiler portability., 23
CC_PACK
Compiler portability., 23
CC_ROMCONST
Compiler portability., 24
CC_SECTION
Compiler portability., 22
CC_UNLIKELY
Compiler portability., 24
CC_USED
Compiler portability., 22
CC_WEAK
Compiler portability., 22
ccportab.h, 563
ch.h, 592
ch0
System Management, 124
ch1
System Management, 125
ch_binary_semaphore, 469
sem, 470

CH_CFG_CONTEXT_SWITCH_HOOK
Options, 54
CH_CFG_FACTORY_GENERIC_BUFFERS
Dynamic Objects Factory, 443
Options, 48
CH_CFG_FACTORY_MAILBOXES
Dynamic Objects Factory, 443, 444
Options, 48
CH_CFG_FACTORY_MAX_NAMES_LENGTH
Dynamic Objects Factory, 443
Options, 48
CH_CFG_FACTORY_OBJ_FIFOS
Dynamic Objects Factory, 444
Options, 49
CH_CFG_FACTORY_OBJECTS_REGISTRY
Dynamic Objects Factory, 443
Options, 48
CH_CFG_FACTORY_PIPES
Dynamic Objects Factory, 444
Options, 49
CH_CFG_FACTORY_SEMAPHORES
Dynamic Objects Factory, 443
Options, 48
CH_CFG_IDLE_ENTER_HOOK
Options, 54
CH_CFG_IDLE_LEAVE_HOOK
Options, 55
CH_CFG_IDLE_LOOP_HOOK
Options, 55
CH_CFG_INTERVALS_SIZE
Options, 40
CH_CFG_IRQ_EPILOGUE_HOOK
Options, 54
CH_CFG_IRQ_PROLOGUE_HOOK
Options, 54
CH_CFG_MEMCORE_SIZE
Core Memory Manager, 377
Options, 46
CH_CFG_NO_IDLE_THREAD
Options, 41
CH_CFG_OPTIMIZE_SPEED
Options, 41
CH_CFG_OS_INSTANCE_EXTRA_FIELDS
Options, 51
CH_CFG_OS_INSTANCE_INIT_HOOK
Options, 52
CH_CFG_RUNTIME_FAULTS_HOOK
Options, 56
CH_CFG_SMP_MODE
Options, 39
CH_CFG_ST_FREQUENCY
Options, 40
CH_CFG_ST_RESOLUTION
Options, 40
CH_CFG_ST_TIMEDELTA
Options, 40
Restrictions, 57
CH_CFG_SYSTEM_EXTRA_FIELDS
Options, 51
CH_CFG_SYSTEM_HALT_HOOK
Options, 55
CH_CFG_SYSTEM_INIT_HOOK
Options, 51
CH_CFG_SYSTEM_TICK_HOOK
Options, 55
CH_CFG_THREAD_EXIT_HOOK
Options, 53
CH_CFG_THREAD_EXTRA_FIELDS
Options, 53
CH_CFG_THREAD_INIT_HOOK
Options, 53
CH_CFG_TIME_QUANTUM
Options, 41
CH_CFG_TIME_TYPES_SIZE
Options, 40
CH_CFG_TRACE_HOOK
Options, 56
CH_CFG_USE_CONDVARS
Options, 43
Restrictions, 57
CH_CFG_USE_CONDVARS_TIMEOUT
Options, 44
CH_CFG_USE_DELEGATES
Options, 47
Version Numbers and Identification, 316
CH_CFG_USE_DYNAMIC
Options, 45
Restrictions, 57
CH_CFG_USE_EVENTS
Options, 44
CH_CFG_USE_EVENTS_TIMEOUT
Options, 44
CH_CFG_USE_FACTORY
Options, 48
Version Numbers and Identification, 315
CH_CFG_USE_HEAP
Options, 46
Version Numbers and Identification, 316
CH_CFG_USE_JOBS
Options, 47
Version Numbers and Identification, 316
CH_CFG_USE_MAILBOXES
Options, 45
Version Numbers and Identification, 316
CH_CFG_USE_MEMCORE
Options, 45
CH_CFG_USE_MEMPOOLS
Options, 46
Version Numbers and Identification, 316
CH_CFG_USE_MESSAGES
Options, 44
CH_CFG_USE_MESSAGES_PRIORITY
Options, 45
CH_CFG_USE_MUTEXES
Options, 43
Restrictions, 57

CH_CFG_USE_MUTEXES_RECURSIVE
Options, 43
CH_CFG_USE_OBJ_CACHES
Options, 47
Version Numbers and Identification, 316
CH_CFG_USE_OBJ_FIFOS
Options, 46
Version Numbers and Identification, 316
CH_CFG_USE_PIPES
Options, 47
Version Numbers and Identification, 316
CH_CFG_USE_REGISTRY
Options, 42
CH_CFG_USE_SEMAPHORES
Options, 42
CH_CFG_USE_SEMAPHORES_PRIORITY
Options, 43
CH_CFG_USE_TIMESTAMP
Options, 42
CH_CFG_USE_TM
Options, 41
Restrictions, 57
CH_CFG_USE_WAITEXIT
Options, 42
ch_core0_cfg
System Management, 125
ch_core1_cfg
System Management, 125
CH_CUSTOMER_ID_CODE
Customer Information, 27
CH_CUSTOMER_ID_STRING
Customer Information, 27
CH_CUSTOMER_LIC_EX
Customer Information, 29
CH_CUSTOMER_LIC_NIL
Customer Information, 28
CH_CUSTOMER_LIC_OSLIB
Customer Information, 28
CH_CUSTOMER_LIC_PORT_ARM79
Customer Information, 29
CH_CUSTOMER_LIC_PORT_CM0
Customer Information, 29
CH_CUSTOMER_LIC_PORT_CM3
Customer Information, 29
CH_CUSTOMER_LIC_PORT_CM33
Customer Information, 29
CH_CUSTOMER_LIC_PORT_CM4
Customer Information, 29
CH_CUSTOMER_LIC_PORT_CM7
Customer Information, 29
CH_CUSTOMER_LIC_PORT_E200Z0
Customer Information, 29
CH_CUSTOMER_LIC_PORT_E200Z2
Customer Information, 29
CH_CUSTOMER_LIC_PORT_E200Z3
Customer Information, 30
CH_CUSTOMER_LIC_PORT_E200Z4
Customer Information, 30
CH_CUSTOMER_LIC_RT
Customer Information, 28
CH_CUSTOMER_LIC_SB
Customer Information, 29
CH_CUSTOMER_LICENSE_EOS_DATE
Customer Information, 28
CH_CUSTOMER_LICENSE_VERSION_DATE
Customer Information, 30
CH_CUSTOMER_LICENSE_VERSION_MONTH
Customer Information, 28
CH_CUSTOMER_LICENSE_VERSION_YEAR
Customer Information, 28
CH_DBG_ENABLE_ASSERTS
Options, 49
CH_DBG_ENABLE_CHECKS
Options, 49
CH_DBG_ENABLE_STACK_CHECK
Options, 50
CH_DBG_FILL_THREADS
Options, 50
CH_DBG_STACK_FILL_VALUE
Checks and Assertions, 292
CH_DBG_STATISTICS
Options, 49
Restrictions, 57
CH_DBG_SYSTEM_STATE_CHECK
Options, 49
CH_DBG_THREADS_PROFILING
Options, 51
CH_DBG_TRACE_BUFFER_SIZE
Options, 50
Tracing, 303
CH_DBG_TRACE_MASK
Options, 50
Restrictions, 57
Tracing, 303
CH_DBG_TRACE_MASK_ALL
Tracing, 303
CH_DBG_TRACE_MASK_DISABLED
Tracing, 302
CH_DBG_TRACE_MASK_HALT
Tracing, 302
CH_DBG_TRACE_MASK_ISR
Tracing, 302
CH_DBG_TRACE_MASK_NONE
Tracing, 302
CH_DBG_TRACE_MASK_READY
Tracing, 302
CH_DBG_TRACE_MASK_SLOW
Tracing, 302
CH_DBG_TRACE_MASK_SWITCH
Tracing, 302
CH_DBG_TRACE_MASK_USER
Tracing, 302
ch_debug
Registry, 288
ch_delta_list, 471
delta, 471

next, 471
prev, 471
ch_delta_list_t
Lists and Queues, 75
CH_DEPLOY_NONE
License Settings, 31
CH_DEPLOY_UNLIMITED
License Settings, 31
ch_dlist_dequeue
Lists and Queues, 84
ch_dlist_init
Lists and Queues, 81
ch_dlist_insert
Lists and Queues, 83
ch_dlist_insert_after
Lists and Queues, 82
ch_dlist_insert_before
Lists and Queues, 83
ch_dlist_isempty
Lists and Queues, 81
ch_dlist_isfirst
Lists and Queues, 82
ch_dlist_islast
Lists and Queues, 82
ch_dlist_notempty
Lists and Queues, 81
ch_dlist_remove_first
Lists and Queues, 84
ch_dyn_element, 472
name, 473
next, 472
refs, 472
ch_dyn_list, 473
next, 474
ch_dyn_mailbox, 474
element, 475
mbx, 475
ch_dyn_object, 475
element, 476
ch_dyn_objects_fifo, 476
element, 477
fifo, 477
ch_dyn_pipe, 477
element, 479
pipe, 479
ch_dyn_semaphore, 479
element, 480
sem, 480
ch_factory
Dynamic Objects Factory, 467
CH_FACTORY_REQUIRES_HEAP
Dynamic Objects Factory, 445
CH_FACTORY_REQUIRES_POOLS
Dynamic Objects Factory, 444
CH_FAST_IRQ_HANDLER
System Management, 108
CH_FEATURES_BASIC
License Settings, 31
CH_FEATURES_FULL
License Settings, 31
CH_FEATURES_INTERMEDIATE
License Settings, 31
CH_FLAG_MODE_HEAP
Scheduler, 90
CH_FLAG_MODE_MASK
Scheduler, 90
CH_FLAG_MODE_MPOOL
Scheduler, 90
CH_FLAG_MODE_STATIC
Scheduler, 90
CH_FLAG_TERMINATE
Scheduler, 91
CH_HEAP_ALIGNMENT
Memory Heaps, 387
CH_HEAP_AREA
Memory Heaps, 387
CH_INTEGRITY_PORT
System Management, 106
CH_INTEGRITY_REGISTRY
System Management, 106
CH_INTEGRITY_RLIST
System Management, 106
CH_INTEGRITY_VTLIST
System Management, 106
CH_IRQ_EPILOGUE
System Management, 107
CH_IRQ_HANDLER
System Management, 108
CH_IRQ_IS_VALID_KERNEL_PRIORITY
System Management, 107
CH_IRQ_IS_VALID_PRIORITY
System Management, 106
CH_IRQ_PROLOGUE
System Management, 107
ch_job_descriptor, 480
jobarg, 481
jobfunc, 481
ch_jobs_queue, 481
free, 482
mbx, 482
CH_KERNEL_MAJOR
Version Numbers and Identification, 35
CH_KERNEL_MINOR
Version Numbers and Identification, 35
CH_KERNEL_PATCH
Version Numbers and Identification, 35
CH_KERNEL_STABLE
Version Numbers and Identification, 34
CH_KERNEL_VERSION
Version Numbers and Identification, 34
CH_LICENSE
Customer Information, 28
CH_LICENSE_COMMERCIAL_DEV_1000
License Settings, 32
CH_LICENSE_COMMERCIAL_DEV_5000
License Settings, 32

CH_LICENSE_COMMERCIAL_FREE
License Settings, 31
CH_LICENSE_COMMERCIAL_FULL
License Settings, 32
CH_LICENSE_COMMERCIAL_RUNTIME
License Settings, 32
CH_LICENSE_FEATURES
License Settings, 33
CH_LICENSE_GPL
License Settings, 31
CH_LICENSE_GPL_EXCEPTION
License Settings, 31
CH_LICENSE_ID_CODE
License Settings, 32
CH_LICENSE_ID_STRING
License Settings, 32
CH_LICENSE_MAX_DEPLOY
License Settings, 33
CH_LICENSE_MODIFIABLE_CODE
License Settings, 33
CH_LICENSE_PARTNER
License Settings, 32
CH_LICENSE_TYPE_STRING
License Settings, 32
ch_list, 483
next, 483
ch_list_init
Lists and Queues, 75
ch_list_isempty
Lists and Queues, 75
ch_list_link
Lists and Queues, 76
ch_list_notempty
Lists and Queues, 76
ch_list_t
Lists and Queues, 75
ch_list_unlink
Lists and Queues, 76
ch_memcore
Core Memory Manager, 384
ch_mutex, 484
cnt, 485
next, 485
owner, 485
queue, 485
ch_objects_cache, 485
cache_sem, 488
hashn, 487
hashp, 487
lru, 488
lru_sem, 488
objn, 487
objsz, 487
objvp, 487
readf, 488
writef, 488
ch_objects_factory, 489
buf_list, 490
fifo_list, 491
mbx_list, 491
mtx, 490
obj_list, 490
obj_pool, 490
pipe_list, 491
sem_list, 490
sem_pool, 490
ch_objects_fifo, 491
free, 492
mbx, 492
ch_oc_hash_header, 493
hash_next, 494
hash_prev, 494
ch_oc_lru_header, 494
hash_next, 496
hash_prev, 496
lru_next, 496
lru_prev, 496
ch_oc_object, 496
dptr, 499
hash_next, 498
hash_prev, 498
lru_next, 498
lru_prev, 498
obj_flags, 499
obj_group, 498
obj_key, 498
obj_sem, 498
ch_os_instance, 499
config, 502
core_id, 501
dbg, 502
kernel_stats, 502
mainthread, 502
reglist, 501
rfcu, 501
rlist, 501
trace_buffer, 502
vtlist, 501
ch_os_instance_config, 503
idlethread_base, 504
idlethread_end, 504
mainthread_base, 503
mainthread_end, 504
name, 503
CH_OSLIB_MAJOR
Version Numbers and Identification, 315
CH_OSLIB_MINOR
Version Numbers and Identification, 315
CH_OSLIB_PATCH
Version Numbers and Identification, 315
CH_OSLIB_STABLE
Version Numbers and Identification, 315
CH_OSLIB_VERSION
Version Numbers and Identification, 315
CH_PORT_SUPPORTS_RECURSIVE_LOCKS
Port Interface, 58

ch_pqueue_init
Lists and Queues, 79
ch_pqueue_insert_ahead
Lists and Queues, 80
ch_pqueue_insert_behind
Lists and Queues, 80
ch_pqueue_remove_highest
Lists and Queues, 80
ch_priority_queue, 504
next, 505
prev, 505
prio, 505
ch_priority_queue_t
Lists and Queues, 75
ch_queue, 505
next, 506
prev, 506
CH_QUEUE_DECL
Lists and Queues, 74
ch_queue_dequeue
Lists and Queues, 79
ch_queue_fifo_remove
Lists and Queues, 78
ch_queue_init
Lists and Queues, 77
ch_queue_insert
Lists and Queues, 78
ch_queue_isempty
Lists and Queues, 77
ch_queue_lifo_remove
Lists and Queues, 78
ch_queue_notempty
Lists and Queues, 77
ch_queue_t
Lists and Queues, 75
ch_ready_list, 506
current, 507
pqueue, 507
ch_registered_static_object, 508
element, 509
objp, 509
ch_registry, 509
queue, 510
ch_rfcu, 510
mask, 510
CH_RFCU_ALL_FAULTS
Runtime Faults Collection Unit, 70
CH_RFCU_VT_INSUFFICIENT_DELTA
Runtime Faults Collection Unit, 70
CH_RFCU_VT_SKIPPED_DEADLINE
Runtime Faults Collection Unit, 70
ch_sch_prio_insert
Scheduler, 94
ch_semaphore, 511
cnt, 512
queue, 512
CH_STATE_CURRENT
Scheduler, 88
CH_STATE_FINAL
Scheduler, 90
CH_STATE_NAMES
Scheduler, 90
CH_STATE_QUEUED
Scheduler, 88
CH_STATE_READY
Scheduler, 88
CH_STATE_SLEEPING
Scheduler, 89
CH_STATE_SNDMSG
Scheduler, 89
CH_STATE_SNDMSGQ
Scheduler, 89
CH_STATE_SUSPENDED
Scheduler, 88
CH_STATE_WTANDEVT
Scheduler, 89
CH_STATE_WTCOND
Scheduler, 89
CH_STATE_WTEXIT
Scheduler, 89
CH_STATE_WTMSG
Scheduler, 90
CH_STATE_WTMTX
Scheduler, 89
CH_STATE_WTOREVT
Scheduler, 89
CH_STATE_WTSEM
Scheduler, 88
CH_STATE_WTSTART
Scheduler, 88
CH_SYS_CORE0_MEMORY
System Management, 106
CH_SYS_CORE1_MEMORY
System Management, 106
ch_sys_halted
OS Types and Structures, 67
ch_sys_initializing
OS Types and Structures, 67
ch_sys_running
OS Types and Structures, 67
ch_sys_uninit
OS Types and Structures, 67
ch_system, 512
instances, 513
reglist, 513
rfcu, 513
state, 513
System Management, 124
tmc, 513
ch_system_debug, 514
isr_cnt, 515
lock_cnt, 515
panic_msg, 515
ch_system_t
OS Types and Structures, 66
ch_thread, 515

ctx, 517
epending, 521
ewmask, 520
exitcode, 519
flags, 518
hdr, 517
list, 517
mpool, 522
msgqueue, 521
mtxlist, 521
name, 518
owner, 518
pqueue, 517
queue, 517
rdymsg, 519
realprio, 521
refs, 518
rqueue, 518
sentmsg, 520
state, 518
stats, 522
ticks, 519
time, 519
u, 521
wabase, 518
waiting, 521
wtmtxp, 520
wtobjp, 519
wtsemp, 520
wttrp, 520
ch_threads_queue, 522
queue, 523
CH_TRACE_TYPE_HALT
Tracing, 302
CH_TRACE_TYPE_ISR_ENTER
Tracing, 301
CH_TRACE_TYPE_ISR_LEAVE
Tracing, 301
CH_TRACE_TYPE_READY
Tracing, 301
CH_TRACE_TYPE_SWITCH
Tracing, 301
CH_TRACE_TYPE_UNUSED
Tracing, 301
CH_TRACE_TYPE_USER
Tracing, 302
CH_VERSION
Release Information, 26
CH_VERSION_DATE
Release Information, 26
CH_VERSION_MONTH
Release Information, 26
CH_VERSION_NICKNAME
Release Information, 26
CH_VERSION_PATCH
Release Information, 26
CH_VERSION_STABLE
Release Information, 25
CH_VERSION_YEAR
Release Information, 26
ch_virtual_timer, 523
dlist, 524
func, 524
par, 524
reload, 524
ch_virtual_timers_list, 524
dlist, 525
laststamp, 526
lasttime, 526
systime, 525
chalign.h, 593
chbsem.h, 566
chBSemGetStateI
Binary Semaphores, 328
chBSemObjectInit
Binary Semaphores, 320
chBSemReset
Binary Semaphores, 325
chBSemResetI
Binary Semaphores, 324
chBSemSignal
Binary Semaphores, 326
chBSemSignalI
Binary Semaphores, 326
chBSemWait
Binary Semaphores, 321
chBSemWaitS
Binary Semaphores, 321
chBSemWaitTimeout
Binary Semaphores, 323
chBSemWaitTimeoutS
Binary Semaphores, 322
chCacheGetObject
Objects Caches, 435
chCacheObjectInit
Objects Caches, 434
chCacheReadObject
Objects Caches, 437
chCacheReleaseObject
Objects Caches, 438
chCacheReleaseObjectI
Objects Caches, 436
chCacheWriteObject
Objects Caches, 437
chchecks.h, 594
chcond.c, 622
chcond.h, 594
chCondBroadcast
Condition Variables, 237
chCondBroadcastI
Condition Variables, 237
chCondObjectInit
Condition Variables, 235
chCondSignal
Condition Variables, 235
chCondSignalI

Condition Variables, 236
chCondWait
Condition Variables, 238
chCondWaitS
Condition Variables, 239
chCondWaitTimeout
Condition Variables, 240
chCondWaitTimeoutS
Condition Variables, 241
chconf.h, 634
chCoreAlloc
Core Memory Manager, 383
chCoreAllocAligned
Core Memory Manager, 382
chCoreAllocAlignedI
Core Memory Manager, 381
chCoreAllocAlignedWithOffset
Core Memory Manager, 377
chCoreAllocAlignedWithOffsetI
Core Memory Manager, 377
chCoreAllocFromBase
Core Memory Manager, 379
chCoreAllocFromBaseI
Core Memory Manager, 378
chCoreAllocFromTop
Core Memory Manager, 380
chCoreAllocFromTopI
Core Memory Manager, 379
chCoreAllocI
Core Memory Manager, 382
chCoreGetStatusX
Core Memory Manager, 381
chcustomer.h, 564
chDbgAssert
Checks and Assertions, 293
chDbgCheck
Checks and Assertions, 292
chDbgCheckClassI
Checks and Assertions, 292, 298
chDbgCheckClassS
Checks and Assertions, 292, 298
chDbgWriteTrace
Tracing, 304
chDbgWriteTraceI
Tracing, 304
chdebug.c, 622
chdebug.h, 595
chdebug_t, 526
identifier, 528
instancesnum, 531
intctxsize, 530
intervalsize, 530
off_ctx, 529
off_flags, 530
off_inst_core_id, 532
off_inst_reglist, 532
off_inst_rfcu, 532
off_inst_rlist, 531
off_inst_rlist_current, 531
off_inst_vtlist, 532
off_name, 529
off_newer, 529
off_older, 529
off_preempt, 530
off_prio, 529
off_refs, 530
off_reserved, 530
off_state, 530
off_stklimit, 529
off_sys_instances, 531
off_sys_reglist, 531
off_sys_reserved, 531
off_sys_rfcu, 531
off_sys_state, 531
off_time, 530
ptrsize, 528
size, 528
threadsize, 529
timesize, 529
version, 528
zero, 528
chDelegateCallDirect0
Delegate Threads, 359
chDelegateCallDirect1
Delegate Threads, 360
chDelegateCallDirect2
Delegate Threads, 360
chDelegateCallDirect3
Delegate Threads, 361
chDelegateCallDirect4
Delegate Threads, 362
chDelegateCallVeneer
Delegate Threads, 357
chDelegateDispatch
Delegate Threads, 358
chDelegateDispatchTimeout
Delegate Threads, 358
chdelegates.c, 583
chdelegates.h, 568
chdynamic.c, 623
chdynamic.h, 596
chearly.h, 596
Checks, 56
Checks and Assertions, 289
__dbg_check_disable, 291, 294
__dbg_check_enable, 291, 294
__dbg_check_enter_isr, 292, 297
__dbg_check_leave_isr, 292, 297
__dbg_check_lock, 291, 295
__dbg_check_lock_from_isr, 292, 296
__dbg_check_suspend, 291, 294
__dbg_check_unlock, 291, 295
__dbg_check_unlock_from_isr, 292, 296
__dbg_object_init, 299
CH_DBG_STACK_FILL_VALUE, 292
chDbgAssert, 293

chDbgCheck, 292
chDbgCheckClassI, 292, 298
chDbgCheckClassS, 292, 298
system_debug_t, 293
chevents.c, 624
chevents.h, 597
chEvtAddEvents
Event Flags, 251
chEvtAddEventsI
Event Flags, 265
chEvtBroadcast
Event Flags, 264
chEvtBroadcastFlags
Event Flags, 255
chEvtBroadcastFlagsI
Event Flags, 254
chEvtBroadcastI
Event Flags, 265
chEvtDispatch
Event Flags, 255
chEvtGetAndClearEvents
Event Flags, 250
chEvtGetAndClearEventsI
Event Flags, 249
chEvtGetAndClearFlags
Event Flags, 252
chEvtGetAndClearFlagsI
Event Flags, 251
chEvtGetEventsX
Event Flags, 266
chEvtIsListeningI
Event Flags, 264
chEvtObjectInit
Event Flags, 262
chEvtRegister
Event Flags, 263
chEvtRegisterMask
Event Flags, 262
chEvtRegisterMaskWithFlags
Event Flags, 248
chEvtRegisterMaskWithFlagsI
Event Flags, 247
chEvtSignal
Event Flags, 253
chEvtSignalI
Event Flags, 252
chEvtUnregister
Event Flags, 249
chEvtWaitAll
Event Flags, 247, 258
chEvtWaitAllTimeout
Event Flags, 261
chEvtWaitAny
Event Flags, 247, 257
chEvtWaitAnyTimeout
Event Flags, 260
chEvtWaitOne
Event Flags, 246, 256
chEvtWaitOneTimeout
Event Flags, 259
chfactory.c, 584
chfactory.h, 569
chFactoryCreateBuffer
Dynamic Objects Factory, 453
chFactoryCreateMailbox
Dynamic Objects Factory, 457
chFactoryCreateObjectsFIFO
Dynamic Objects Factory, 459
chFactoryCreatePipe
Dynamic Objects Factory, 462
chFactoryCreateSemaphore
Dynamic Objects Factory, 455
chFactoryDuplicateReference
Dynamic Objects Factory, 464
chFactoryFindBuffer
Dynamic Objects Factory, 454
chFactoryFindMailbox
Dynamic Objects Factory, 458
chFactoryFindObject
Dynamic Objects Factory, 451
chFactoryFindObjectByPointer
Dynamic Objects Factory, 451
chFactoryFindObjectsFIFO
Dynamic Objects Factory, 460
chFactoryFindPipe
Dynamic Objects Factory, 463
chFactoryFindSemaphore
Dynamic Objects Factory, 456
chFactoryGetBuffer
Dynamic Objects Factory, 465
chFactoryGetBufferSize
Dynamic Objects Factory, 465
chFactoryGetMailbox
Dynamic Objects Factory, 466
chFactoryGetObject
Dynamic Objects Factory, 464
chFactoryGetObjectsFIFO
Dynamic Objects Factory, 466
chFactoryGetPipe
Dynamic Objects Factory, 467
chFactoryGetSemaphore
Dynamic Objects Factory, 466
chFactoryRegisterObject
Dynamic Objects Factory, 450
chFactoryReleaseBuffer
Dynamic Objects Factory, 454
chFactoryReleaseMailbox
Dynamic Objects Factory, 459
chFactoryReleaseObject
Dynamic Objects Factory, 452
chFactoryReleaseObjectsFIFO
Dynamic Objects Factory, 461
chFactoryReleasePipe
Dynamic Objects Factory, 463
chFactoryReleaseSemaphore
Dynamic Objects Factory, 457

chFifoObjectInit
Objects FIFOs, 415
chFifoObjectInitAligned
Objects FIFOs, 414
chFifoReceiveObjectI
Objects FIFOs, 424
chFifoReceiveObjectTimeout
Objects FIFOs, 426
chFifoReceiveObjectTimeoutS
Objects FIFOs, 425
chFifoReturnObject
Objects FIFOs, 419
chFifoReturnObjectI
Objects FIFOs, 418
chFifoReturnObjectS
Objects FIFOs, 419
chFifoSendObject
Objects FIFOs, 421
chFifoSendObjectAhead
Objects FIFOs, 423
chFifoSendObjectAheadI
Objects FIFOs, 422
chFifoSendObjectAheadS
Objects FIFOs, 423
chFifoSendObjectI
Objects FIFOs, 420
chFifoSendObjectS
Objects FIFOs, 421
chFifoTakeObjectI
Objects FIFOs, 416
chFifoTakeObjectTimeout
Objects FIFOs, 417
chFifoTakeObjectTimeoutS
Objects FIFOs, 416
chGuardedPoolAdd
Memory Pools, 410
chGuardedPoolAddI
Memory Pools, 411
chGuardedPoolAddS
Memory Pools, 412
chGuardedPoolAllocI
Memory Pools, 407
chGuardedPoolAllocTimeout
Memory Pools, 402
chGuardedPoolAllocTimeoutS
Memory Pools, 401
chGuardedPoolFree
Memory Pools, 403
chGuardedPoolFreeI
Memory Pools, 408
chGuardedPoolFreeS
Memory Pools, 409
chGuardedPoolGetCounterI
Memory Pools, 407
chGuardedPoolLoadArray
Memory Pools, 400
chGuardedPoolObjectInit
Memory Pools, 406
chGuardedPoolObjectInitAligned
Memory Pools, 399
chHeapAlloc
Memory Heaps, 390
chHeapAllocAligned
Memory Heaps, 389
chHeapFree
Memory Heaps, 389
chHeapGetSize
Memory Heaps, 391
chHeapObjectInit
Memory Heaps, 388
chHeapStatus
Memory Heaps, 390
chInstanceObjectInit
OS Instances, 69
chinstances.c, 625
chinstances.h, 599
chJobDispatch
Jobs Queues, 373
chJobDispatchTimeout
Jobs Queues, 374
chJobGet
Jobs Queues, 365
chJobGetI
Jobs Queues, 366
chJobGetTimeout
Jobs Queues, 368
chJobGetTimeoutS
Jobs Queues, 367
chJobObjectInit
Jobs Queues, 365
chJobPost
Jobs Queues, 370
chJobPostAhead
Jobs Queues, 372
chJobPostAheadI
Jobs Queues, 371
chJobPostAheadS
Jobs Queues, 371
chJobPostI
Jobs Queues, 368
chJobPostS
Jobs Queues, 369
chjobs.h, 572
chlib.h, 574
chlicense.h, 565
chlists.h, 600
chMBFetchI
Mailboxes, 341
chMBFetchTimeout
Mailboxes, 339
chMBFetchTimeoutS
Mailboxes, 340
chMBGetFreeCountI
Mailboxes, 342
chMBGetSizeI
Mailboxes, 342

chMBGetUsedCountI
Mailboxes, 342
chMBObjectInit
Mailboxes, 331
chmboxes.c, 586
chmboxes.h, 575
chMBPeekI
Mailboxes, 343
chMBPostAheadI
Mailboxes, 338
chMBPostAheadTimeout
Mailboxes, 336
chMBPostAheadTimeoutS
Mailboxes, 337
chMBPostI
Mailboxes, 335
chMBPostTimeout
Mailboxes, 333
chMBPostTimeoutS
Mailboxes, 334
chMBReset
Mailboxes, 332
chMBResetI
Mailboxes, 332
chMBResumeX
Mailboxes, 343
chmemcore.c, 587
chmemcore.h, 576
chmemheaps.c, 587
chmemheaps.h, 577
chmempools.c, 588
chmempools.h, 578
chmsg.c, 625
chmsg.h, 602
chMsgGet
Synchronous Messages, 276
chMsgIsPendingI
Synchronous Messages, 275
chMsgPoll
Synchronous Messages, 274
chMsgPollS
Synchronous Messages, 270
chMsgRelease
Synchronous Messages, 271
chMsgReleaseS
Synchronous Messages, 276
chMsgSend
Synchronous Messages, 268
chMsgWait
Synchronous Messages, 272
chMsgWaitS
Synchronous Messages, 268
chMsgWaitTimeout
Synchronous Messages, 273
chMsgWaitTimeoutS
Synchronous Messages, 269
chmtx.c, 626
chmtx.h, 602
chMtxGetNextMutexX
Mutexes, 232
chMtxGetOwnerI
Mutexes, 232
chMtxLock
Mutexes, 224
chMtxLockS
Mutexes, 225
chMtxObjectInit
Mutexes, 224
chMtxQueueNotEmptyS
Mutexes, 231
chMtxTryLock
Mutexes, 226
chMtxTryLockS
Mutexes, 227
chMtxUnlock
Mutexes, 228
chMtxUnlockAll
Mutexes, 231
chMtxUnlockAllS
Mutexes, 230
chMtxUnlockS
Mutexes, 229
chobjcaches.c, 589
chobjcaches.h, 580
chobjects.h, 604
chobjfifos.h, 581
chPipeGetFreeCount
Pipes, 352
chPipeGetSize
Pipes, 352
chPipeGetUsedCount
Pipes, 352
chPipeObjectInit
Pipes, 349
chPipeReadTimeout
Pipes, 351
chPipeReset
Pipes, 349
chPipeResume
Pipes, 353
chpipes.c, 591
chpipes.h, 582
chPipeWriteTimeout
Pipes, 350
chPoolAdd
Memory Pools, 404
chPoolAddI
Memory Pools, 405
chPoolAlloc
Memory Pools, 397
chPoolAllocI
Memory Pools, 396
chPoolFree
Memory Pools, 398
chPoolFreeI
Memory Pools, 398

chPoolLoadArray
Memory Pools, 396
chPoolObjectInit
Memory Pools, 404
chPoolObjectInitAligned
Memory Pools, 395
chport.h, 605
chRegFindThreadByName
Registry, 284
chRegFindThreadByPointer
Registry, 285
chRegFindThreadByWorkingArea
Registry, 285
chRegFirstThread
Registry, 282
chRegGetThreadNameX
Registry, 287
chregistry.c, 626
chregistry.h, 605
chRegNextThread
Registry, 283
chRegSetThreadName
Registry, 287
chRegSetThreadNameX
Registry, 288
chrestrictions.h, 606
chrfcu.c, 627
chrfcu.h, 607
chRFCUCollectFaultsI
Runtime Faults Collection Unit, 71
chRFCUGetAndClearFaultsI
Runtime Faults Collection Unit, 71
chschd.c, 627
chschd.h, 607
chSchDoPreemption
Scheduler, 100
chSchDoYieldS
Scheduler, 101
chSchGoSleepS
Scheduler, 96
chSchGoSleepTimeoutS
Scheduler, 96
chSchIsPreemptionRequired
Scheduler, 100
chSchObjectInit
Scheduler, 102
chSchPreemption
Scheduler, 101
chSchReadyI
Scheduler, 95
chSchRescheduleS
Scheduler, 99
chSchSelectFirst
Scheduler, 102
chSchWakeupS
Scheduler, 98
chsem.c, 628
chsem.h, 610
chSemAddCounterI
Counting Semaphores, 216
chSemFastSignalI
Counting Semaphores, 220
chSemFastWaitI
Counting Semaphores, 220
chSemGetCounterI
Counting Semaphores, 221
chSemObjectInit
Counting Semaphores, 209
chSemReset
Counting Semaphores, 218
chSemResetI
Counting Semaphores, 219
chSemResetWithMessage
Counting Semaphores, 209
chSemResetWithMessageI
Counting Semaphores, 210
chSemSignal
Counting Semaphores, 215
chSemSignalI
Counting Semaphores, 215
chSemSignalWait
Counting Semaphores, 217
chSemWait
Counting Semaphores, 211
chSemWaitS
Counting Semaphores, 212
chSemWaitTimeout
Counting Semaphores, 213
chSemWaitTimeoutS
Counting Semaphores, 214
chstats.c, 629
chstats.h, 611
chsys.c, 630
chsys.h, 611
chSysDisable
System Management, 120
chSysEnable
System Management, 120
chSysGetIdleThreadX
System Management, 115
chSysGetRealtimeCounterX
System Management, 112
chSysGetStatusAndLockX
System Management, 117
chSysHalt
OS Types and Structures, 67
System Management, 114
chSysInit
System Management, 113
chSysIntegrityCheckI
System Management, 115
chSysIsCounterWithinX
System Management, 118
chSysLock
System Management, 121
chSysLockFromISR

System Management, 122
chSysNotifyInstance
System Management, 124
chSysPolledDelayX
System Management, 119
chSysRestoreStatusX
System Management, 118
chSysSuspend
System Management, 120
chSysSwitch
System Management, 112
chSysTimerHandlerI
System Management, 116
chSysUnconditionalLock
System Management, 123
chSysUnconditionalUnlock
System Management, 123
chSysUnlock
System Management, 121
chSysUnlockFromISR
System Management, 122
chSysWaitSystemState
System Management, 113
chThdAddRef
Threads, 181
chThdCreate
Threads, 178
chThdCreateFromHeap
Dynamic Threads, 277
chThdCreateFromMemoryPool
Dynamic Threads, 278
chThdCreateI
Threads, 177
chThdCreateStatic
Threads, 179
chThdCreateSuspended
Threads, 176
chThdCreateSuspendedI
Threads, 175
chThdDequeueAllI
Threads, 196
chThdDequeueNextI
Threads, 195
chThdDoDequeueNextI
Threads, 201
chThdEnqueueTimeoutS
Threads, 194
chThdExit
Threads, 183
chThdExitS
Threads, 184
chThdGetPriorityX
Threads, 197
chThdGetSelfX
Threads, 196
chThdGetTicksX
Threads, 197
chThdGetWorkingAreaX
Threads, 198
chThdQueueIsEmptyI
Threads, 201
chThdQueueObjectInit
Threads, 200
chThdRelease
Threads, 182
chThdResume
Threads, 194
chThdResumeI
Threads, 192
chThdResumeS
Threads, 193
chThdSetPriority
Threads, 186
chThdShouldTerminateX
Threads, 198
chThdSleep
Threads, 187
chThdSleepMicroseconds
Threads, 173
chThdSleepMilliseconds
Threads, 172
chThdSleepS
Threads, 199
chThdSleepSeconds
Threads, 172
chThdSleepUntil
Threads, 188
chThdSleepUntilWindowed
Threads, 189
chThdStart
Threads, 180
chThdStartI
Threads, 199
chThdSuspendS
Threads, 190
chThdSuspendTimeoutS
Threads, 191
chThdTerminate
Threads, 187
chThdTerminatedX
Threads, 198
chThdWait
Threads, 185
chThdYield
Threads, 190
chthreads.c, 631
chthreads.h, 613
chtime.h, 616
chTimeAddX
Time and Intervals, 138
chTimeDiffX
Time and Intervals, 138
chTimeI2MS
Time and Intervals, 137
chTimeI2S
Time and Intervals, 136

chTimeI2US
Time and Intervals, 137
chTimeIsInRangeX
Time and Intervals, 138
chTimeMS2I
Time and Intervals, 135
chTimeS2I
Time and Intervals, 135
chTimeStampAddX
Time and Intervals, 139
chTimeStampDiffX
Time and Intervals, 139
chTimeStampIsInRangeX
Time and Intervals, 140
chTimeUS2I
Time and Intervals, 136
chtm.c, 632
chtm.h, 618
chTMChainMeasurementToX
Time Measurement, 204
chTMObjectInit
Time Measurement, 203
chTMStartMeasurementX
Time Measurement, 204
chTMStopMeasurementX
Time Measurement, 204
chtrace.c, 633
chtrace.h, 618
chTraceResume
Tracing, 310
chTraceResumeI
Tracing, 309
chTraceSuspend
Tracing, 309
chTraceSuspendI
Tracing, 308
chTraceWrite
Tracing, 308
chTraceWriteI
Tracing, 307
chversion.h, 566
chvt.c, 634
chvt.h, 620
chVTDoResetI
Virtual Timers, 147
chVTDoSetContinuousI
Virtual Timers, 146
chVTDoSetI
Virtual Timers, 145
chVTDoTickI
Virtual Timers, 149
chVTGetReloadIntervalX
Virtual Timers, 163
chVTGetRemainingIntervalI
Virtual Timers, 148
chVTGetSystemTime
Virtual Timers, 152
chVTGetSystemTimeX
Virtual Timers, 152
chVTGetTimersStateI
Virtual Timers, 155
chVTGetTimeStamp
Virtual Timers, 164
chVTGetTimeStampI
Virtual Timers, 150
chVTIsArmed
Virtual Timers, 157
chVTIsArmedI
Virtual Timers, 156
chVTIsSystemTimeWithin
Virtual Timers, 155
chVTIsSystemTimeWithinX
Virtual Timers, 154
chVTObjectInit
Virtual Timers, 151
chVTReset
Virtual Timers, 158
chVTResetI
Virtual Timers, 157
chVTResetTimeStamp
Virtual Timers, 164
chVTResetTimeStampI
Virtual Timers, 151
chVTSet
Virtual Timers, 160
chVTSetContinuous
Virtual Timers, 162
chVTSetContinuousI
Virtual Timers, 161
chVTSetI
Virtual Timers, 159
chVTSetReloadIntervalX
Virtual Timers, 163
chVTTimeElapsedSinceX
Virtual Timers, 153
cmtx
pipe_t, 550
cnt
ch_mutex, 485
ch_semaphore, 512
mailbox_t, 542
pipe_t, 549
cnt_t
OS Types and Structures, 64
Compiler portability., 21
CC_ALIGN_CODE, 22
CC_ALIGN_DATA, 22
CC_FORCE_INLINE, 23
CC_LIKELY, 24
CC_NO_INLINE, 23
CC_NO_RETURN, 23
CC_PACK, 23
CC_ROMCONST, 24
CC_SECTION, 22
CC_UNLIKELY, 24
CC_USED, 22

CC_WEAK, 22
Complex Services, 413
concepts.dox, 563
Condition Variables, 233
__CONDVAR_DATA, 234
chCondBroadcast, 237
chCondBroadcastI, 237
chCondObjectInit, 235
chCondSignal, 235
chCondSignalI, 236
chCondWait, 238
chCondWaitS, 239
chCondWaitTimeout, 240
chCondWaitTimeoutS, 241
condition_variable_t, 235
CONDVAR_DECL, 234
condition_variable, 532
queue, 533
condition_variable_t
Condition Variables, 235
CONDVAR_DECL
Condition Variables, 234
config
ch_os_instance, 502
Configuration, 36
copy_name
Dynamic Objects Factory, 446
Core Memory Manager, 375
__core_init, 378
CH_CFG_MEMCORE_SIZE, 377
ch_memcore, 384
chCoreAlloc, 383
chCoreAllocAligned, 382
chCoreAllocAlignedI, 381
chCoreAllocAlignedWithOffset, 377
chCoreAllocAlignedWithOffsetI, 377
chCoreAllocFromBase, 379
chCoreAllocFromBaseI, 378
chCoreAllocFromTop, 380
chCoreAllocFromTopI, 379
chCoreAllocI, 382
chCoreGetStatusX, 381
memgetfunc2_t, 378
memgetfunc_t, 378
core_id
ch_os_instance, 501
core_id_t
OS Types and Structures, 64
Counting Semaphores, 206
__SEMAPHORE_DATA, 208
chSemAddCounterI, 216
chSemFastSignalI, 220
chSemFastWaitI, 220
chSemGetCounterI, 221
chSemObjectInit, 209
chSemReset, 218
chSemResetI, 219
chSemResetWithMessage, 209
chSemResetWithMessageI, 210
chSemSignal, 215
chSemSignalI, 215
chSemSignalWait, 217
chSemWait, 211
chSemWaitS, 212
chSemWaitTimeout, 213
chSemWaitTimeoutS, 214
sem_insert, 208
SEMAPHORE_DECL, 208
semaphore_t, 209
ctx
ch_thread, 517
cumulative
time_measurement_t, 555
currcore
System Management, 106
current
ch_ready_list, 507
Customer Information, 27
CH_CUSTOMER_ID_CODE, 27
CH_CUSTOMER_ID_STRING, 27
CH_CUSTOMER_LIC_EX, 29
CH_CUSTOMER_LIC_NIL, 28
CH_CUSTOMER_LIC_OSLIB, 28
CH_CUSTOMER_LIC_PORT_ARM79, 29
CH_CUSTOMER_LIC_PORT_CM0, 29
CH_CUSTOMER_LIC_PORT_CM3, 29
CH_CUSTOMER_LIC_PORT_CM33, 29
CH_CUSTOMER_LIC_PORT_CM4, 29
CH_CUSTOMER_LIC_PORT_CM7, 29
CH_CUSTOMER_LIC_PORT_E200Z0, 29
CH_CUSTOMER_LIC_PORT_E200Z2, 29
CH_CUSTOMER_LIC_PORT_E200Z3, 30
CH_CUSTOMER_LIC_PORT_E200Z4, 30
CH_CUSTOMER_LIC_RT, 28
CH_CUSTOMER_LIC_SB, 29
CH_CUSTOMER_LICENSE_EOS_DATE, 28
CH_CUSTOMER_LICENSE_VERSION_DATE, 30
CH_CUSTOMER_LICENSE_VERSION_MONTH, 28
CH_CUSTOMER_LICENSE_VERSION_YEAR, 28
CH_LICENSE, 28

dbg
ch_os_instance, 502
Debug, 289
default_heap
Memory Heaps, 392
Delegate Threads, 354
__ch_delegate_fn0, 355
__ch_delegate_fn1, 356
__ch_delegate_fn2, 356
__ch_delegate_fn3, 356
__ch_delegate_fn4, 357
chDelegateCallDirect0, 359
chDelegateCallDirect1, 360
chDelegateCallDirect2, 360
chDelegateCallDirect3, 361

chDelegateCallDirect4, 362
chDelegateCallVeneer, 357
chDelegateDispatch, 358
chDelegateDispatchTimeout, 358
delegate_fn0_t, 355
delegate_fn1_t, 355
delegate_fn2_t, 355
delegate_fn3_t, 355
delegate_fn4_t, 355
delegate_veneer_t, 355
delegate_fn0_t
Delegate Threads, 355
delegate_fn1_t
Delegate Threads, 355
delegate_fn2_t
Delegate Threads, 355
delegate_fn3_t
Delegate Threads, 355
delegate_fn4_t
Delegate Threads, 355
delegate_veneer_t
Delegate Threads, 355
delta
ch_delta_list, 471
dlist
ch_virtual_timer, 524
ch_virtual_timers_list, 525
dptr
ch_oc_object, 499
dyn_buffer_t
Dynamic Objects Factory, 445
dyn_create_object_heap
Dynamic Objects Factory, 447
dyn_create_object_pool
Dynamic Objects Factory, 448
dyn_element_t
Dynamic Objects Factory, 445
dyn_find_object
Dynamic Objects Factory, 449
dyn_list_find
Dynamic Objects Factory, 446
dyn_list_find_prev
Dynamic Objects Factory, 446
dyn_list_init
Dynamic Objects Factory, 446
dyn_list_t
Dynamic Objects Factory, 445
dyn_list_unlink
Dynamic Objects Factory, 447
dyn_mailbox_t
Dynamic Objects Factory, 445
dyn_objects_fifo_t
Dynamic Objects Factory, 446
dyn_pipe_t
Dynamic Objects Factory, 446
dyn_release_object_heap
Dynamic Objects Factory, 447
dyn_release_object_pool
Dynamic Objects Factory, 448
dyn_semaphore_t
Dynamic Objects Factory, 445
Dynamic Objects Factory, 439
__factory_init, 449
CH_CFG_FACTORY_GENERIC_BUFFERS, 443
CH_CFG_FACTORY_MAILBOXES, 443, 444
CH_CFG_FACTORY_MAX_NAMES_LENGTH, 443
CH_CFG_FACTORY_OBJ_FIFOS, 444
CH_CFG_FACTORY_OBJECTS_REGISTRY, 443
CH_CFG_FACTORY_PIPES, 444
CH_CFG_FACTORY_SEMAPHORES, 443
ch_factory, 467
CH_FACTORY_REQUIRES_HEAP, 445
CH_FACTORY_REQUIRES_POOLS, 444
chFactoryCreateBuffer, 453
chFactoryCreateMailbox, 457
chFactoryCreateObjectsFIFO, 459
chFactoryCreatePipe, 462
chFactoryCreateSemaphore, 455
chFactoryDuplicateReference, 464
chFactoryFindBuffer, 454
chFactoryFindMailbox, 458
chFactoryFindObject, 451
chFactoryFindObjectByPointer, 451
chFactoryFindObjectsFIFO, 460
chFactoryFindPipe, 463
chFactoryFindSemaphore, 456
chFactoryGetBuffer, 465
chFactoryGetBufferSize, 465
chFactoryGetMailbox, 466
chFactoryGetObject, 464
chFactoryGetObjectsFIFO, 466
chFactoryGetPipe, 467
chFactoryGetSemaphore, 466
chFactoryRegisterObject, 450
chFactoryReleaseBuffer, 454
chFactoryReleaseMailbox, 459
chFactoryReleaseObject, 452
chFactoryReleaseObjectsFIFO, 461
chFactoryReleasePipe, 463
chFactoryReleaseSemaphore, 457
copy_name, 446
dyn_buffer_t, 445
dyn_create_object_heap, 447
dyn_create_object_pool, 448
dyn_element_t, 445
dyn_find_object, 449
dyn_list_find, 446
dyn_list_find_prev, 446
dyn_list_init, 446
dyn_list_t, 445
dyn_list_unlink, 447
dyn_mailbox_t, 445
dyn_objects_fifo_t, 446
dyn_pipe_t, 446
dyn_release_object_heap, 447

dyn_release_object_pool, 448
dyn_semaphore_t, 445
F_LOCK, 443
F_UNLOCK, 443
objects_factory_t, 446
registered_object_t, 445
Dynamic Threads, 277
chThdCreateFromHeap, 277
chThdCreateFromMemoryPool, 278

element
ch_dyn_mailbox, 475
ch_dyn_object, 476
ch_dyn_objects_fifo, 477
ch_dyn_pipe, 479
ch_dyn_semaphore, 480
ch_registered_static_object, 509
epending
ch_thread, 521
Event Flags, 243
__EVENTSOURCE_DATA, 246
ALL_EVENTS, 246
chEvtAddEvents, 251
chEvtAddEventsI, 265
chEvtBroadcast, 264
chEvtBroadcastFlags, 255
chEvtBroadcastFlagsI, 254
chEvtBroadcastI, 265
chEvtDispatch, 255
chEvtGetAndClearEvents, 250
chEvtGetAndClearEventsI, 249
chEvtGetAndClearFlags, 252
chEvtGetAndClearFlagsI, 251
chEvtGetEventsX, 266
chEvtIsListeningI, 264
chEvtObjectInit, 262
chEvtRegister, 263
chEvtRegisterMask, 262
chEvtRegisterMaskWithFlags, 248
chEvtRegisterMaskWithFlagsI, 247
chEvtSignal, 253
chEvtSignalI, 252
chEvtUnregister, 249
chEvtWaitAll, 247, 258
chEvtWaitAllTimeout, 261
chEvtWaitAny, 247, 257
chEvtWaitAnyTimeout, 260
chEvtWaitOne, 246, 256
chEvtWaitOneTimeout, 259
event_listener_t, 247
EVENT_MASK, 246
event_source_t, 247
EVENTSOURCE_DECL, 246
evhandler_t, 247
event_listener, 533
events, 534
flags, 534
listener, 534
next, 534
wflags, 534
event_listener_t
Event Flags, 247
EVENT_MASK
Event Flags, 246
event_source, 535
next, 536
event_source_t
Event Flags, 247
eventflags_t
OS Types and Structures, 64
eventid_t
OS Types and Structures, 64
eventmask_t
OS Types and Structures, 64
events
event_listener, 534
EVENTSOURCE_DECL
Event Flags, 246
evhandler_t
Event Flags, 247
ewmask
ch_thread, 520
exitcode
ch_thread, 519

F_LOCK
Dynamic Objects Factory, 443
F_UNLOCK
Dynamic Objects Factory, 443
FALSE
Version Numbers and Identification, 35
fifo
ch_dyn_objects_fifo, 477
fifo_list
ch_objects_factory, 491
firstprio
Scheduler, 91
flags
ch_thread, 518
event_listener, 534
free
ch_jobs_queue, 482
ch_objects_fifo, 492
heap_header, 538
func
ch_virtual_timer, 524
funcp
thread_descriptor_t, 553

guarded_memory_pool_t, 536
pool, 537
sem, 537
GUARDEDMEMORYPOOL_DECL
Memory Pools, 395

H_BLOCK
Memory Heaps, 386
H_HEAP

Memory Heaps, 386
H_LIMIT
Memory Heaps, 386
H_LOCK
Memory Heaps, 386
H_NEXT
Memory Heaps, 386
H_PAGES
Memory Heaps, 386
H_SIZE
Memory Heaps, 387
H_UNLOCK
Memory Heaps, 386
halt
trace_event_t, 562
hash_get_s
Objects Caches, 432
HASH_INSERT
Objects Caches, 428
hash_next
ch_oc_hash_header, 494
ch_oc_lru_header, 496
ch_oc_object, 498
hash_prev
ch_oc_hash_header, 494
ch_oc_lru_header, 496
ch_oc_object, 498
HASH_REMOVE
Objects Caches, 429
hashn
ch_objects_cache, 487
hashp
ch_objects_cache, 487
hdr
ch_thread, 517
header
memory_heap, 546
heap
heap_header, 538
heap_header, 537
free, 538
heap, 538
next, 538
pages, 538
size, 538
used, 539
heap_header_t
Memory Heaps, 388
HIGHPRIO
Scheduler, 88

identifier
chdebug_t, 528
IDLEPRIO
Scheduler, 87
idlethread_base
ch_os_instance_config, 504
idlethread_end
ch_os_instance_config, 504
instance
thread_descriptor_t, 553
instances
ch_system, 513
instancesnum
chdebug_t, 531
intctxsize
chdebug_t, 530
intervalsize
chdebug_t, 530
Introduction, 1
isr
trace_event_t, 562
isr_cnt
ch_system_debug, 515

job_descriptor_t
Jobs Queues, 364
job_function_t
Jobs Queues, 364
jobarg
ch_job_descriptor, 481
jobfunc
ch_job_descriptor, 481
Jobs Queues, 363
chJobDispatch, 373
chJobDispatchTimeout, 374
chJobGet, 365
chJobGetI, 366
chJobGetTimeout, 368
chJobGetTimeoutS, 367
chJobObjectInit, 365
chJobPost, 370
chJobPostAhead, 372
chJobPostAheadI, 371
chJobPostAheadS, 371
chJobPostI, 368
chJobPostS, 369
job_descriptor_t, 364
job_function_t, 364
jobs_queue_t, 364
MSG_JOB_NULL, 364
jobs_queue_t
Jobs Queues, 364

Kernel Concepts, 2
kernel_stats
ch_os_instance, 502
kernel_stats_t, 539
m_crit_isr, 540
m_crit_thd, 540
n_ctxswc, 540
n_irq, 540

last
time_measurement_t, 555
laststamp
ch_virtual_timers_list, 526
lasttime

ch_virtual_timers_list, 526
lib.dox, 566
License Settings, 30
CH_DEPLOY_NONE, 31
CH_DEPLOY_UNLIMITED, 31
CH_FEATURES_BASIC, 31
CH_FEATURES_FULL, 31
CH_FEATURES_INTERMEDIATE, 31
CH_LICENSE_COMMERCIAL_DEV_1000, 32
CH_LICENSE_COMMERCIAL_DEV_5000, 32
CH_LICENSE_COMMERCIAL_FREE, 31
CH_LICENSE_COMMERCIAL_FULL, 32
CH_LICENSE_COMMERCIAL_RUNTIME, 32
CH_LICENSE_FEATURES, 33
CH_LICENSE_GPL, 31
CH_LICENSE_GPL_EXCEPTION, 31
CH_LICENSE_ID_CODE, 32
CH_LICENSE_ID_STRING, 32
CH_LICENSE_MAX_DEPLOY, 33
CH_LICENSE_MODIFIABLE_CODE, 33
CH_LICENSE_PARTNER, 32
CH_LICENSE_TYPE_STRING, 32
license.dox, 566
likely
OS Types and Structures, 61
list
ch_thread, 517
listener
event_listener, 534
Lists and Queues, 72
__CH_QUEUE_DATA, 74
ch_delta_list_t, 75
ch_dlist_dequeue, 84
ch_dlist_init, 81
ch_dlist_insert, 83
ch_dlist_insert_after, 82
ch_dlist_insert_before, 83
ch_dlist_isempty, 81
ch_dlist_isfirst, 82
ch_dlist_islast, 82
ch_dlist_notempty, 81
ch_dlist_remove_first, 84
ch_list_init, 75
ch_list_isempty, 75
ch_list_link, 76
ch_list_notempty, 76
ch_list_t, 75
ch_list_unlink, 76
ch_pqueue_init, 79
ch_pqueue_insert_ahead, 80
ch_pqueue_insert_behind, 80
ch_pqueue_remove_highest, 80
ch_priority_queue_t, 75
CH_QUEUE_DECL, 74
ch_queue_dequeue, 79
ch_queue_fifo_remove, 78
ch_queue_init, 77
ch_queue_insert, 78
ch_queue_isempty, 77
ch_queue_lifo_remove, 78
ch_queue_notempty, 77
ch_queue_t, 75
lock_cnt
ch_system_debug, 515
LOWPRIO
Scheduler, 87
lru
ch_objects_cache, 488
lru_get_last_s
Objects Caches, 433
LRU_INSERT_HEAD
Objects Caches, 429
LRU_INSERT_TAIL
Objects Caches, 429
lru_next
ch_oc_lru_header, 496
ch_oc_object, 498
lru_prev
ch_oc_lru_header, 496
ch_oc_object, 498
LRU_REMOVE
Objects Caches, 429
lru_sem
ch_objects_cache, 488

m_crit_isr
kernel_stats_t, 540
m_crit_thd
kernel_stats_t, 540
MAILBOX_DECL
Mailboxes, 331
mailbox_t, 540
buffer, 542
cnt, 542
qr, 543
qw, 542
rdptr, 542
reset, 542
top, 542
wrptr, 542
Mailboxes, 329
__MAILBOX_DATA, 330
chMBFetchI, 341
chMBFetchTimeout, 339
chMBFetchTimeoutS, 340
chMBGetFreeCountI, 342
chMBGetSizeI, 342
chMBGetUsedCountI, 342
chMBObjectInit, 331
chMBPeekI, 343
chMBPostAheadI, 338
chMBPostAheadTimeout, 336
chMBPostAheadTimeoutS, 337
chMBPostI, 335
chMBPostTimeout, 333
chMBPostTimeoutS, 334
chMBReset, 332

chMBResetI, 332
chMBResumeX, 343
MAILBOX_DECL, 331
main.dox, 563
mainthread
ch_os_instance, 502
mainthread_base
ch_os_instance_config, 503
mainthread_end
ch_os_instance_config, 504
mask
ch_rfcu, 510
mbx
ch_dyn_mailbox, 475
ch_jobs_queue, 482
ch_objects_fifo, 492
mbx_list
ch_objects_factory, 491
MEM_ALIGN_MASK
Memory Alignment, 126
MEM_ALIGN_NEXT
Memory Alignment, 127
MEM_ALIGN_PREV
Memory Alignment, 127
MEM_IS_ALIGNED
Memory Alignment, 127
MEM_IS_VALID_ALIGNMENT
Memory Alignment, 128
memcore_t, 543
basemem, 543
topmem, 543
memgetfunc2_t
Core Memory Manager, 378
memgetfunc_t
Core Memory Manager, 378
Memory Alignment, 126
MEM_ALIGN_MASK, 126
MEM_ALIGN_NEXT, 127
MEM_ALIGN_PREV, 127
MEM_IS_ALIGNED, 127
MEM_IS_VALID_ALIGNMENT, 128
Memory Heaps, 384
__heap_init, 388
CH_HEAP_ALIGNMENT, 387
CH_HEAP_AREA, 387
chHeapAlloc, 390
chHeapAllocAligned, 389
chHeapFree, 389
chHeapGetSize, 391
chHeapObjectInit, 388
chHeapStatus, 390
default_heap, 392
H_BLOCK, 386
H_HEAP, 386
H_LIMIT, 386
H_LOCK, 386
H_NEXT, 386
H_PAGES, 386
H_SIZE, 387
H_UNLOCK, 386
heap_header_t, 388
memory_heap_t, 388
NPAGES, 387
Memory Management, 375
Memory Pools, 392
__GUARDEDMEMORYPOOL_DATA, 394
__MEMORYPOOL_DATA, 394
chGuardedPoolAdd, 410
chGuardedPoolAddI, 411
chGuardedPoolAddS, 412
chGuardedPoolAllocI, 407
chGuardedPoolAllocTimeout, 402
chGuardedPoolAllocTimeoutS, 401
chGuardedPoolFree, 403
chGuardedPoolFreeI, 408
chGuardedPoolFreeS, 409
chGuardedPoolGetCounterI, 407
chGuardedPoolLoadArray, 400
chGuardedPoolObjectInit, 406
chGuardedPoolObjectInitAligned, 399
chPoolAdd, 404
chPoolAddI, 405
chPoolAlloc, 397
chPoolAllocI, 396
chPoolFree, 398
chPoolFreeI, 398
chPoolLoadArray, 396
chPoolObjectInit, 404
chPoolObjectInitAligned, 395
GUARDEDMEMORYPOOL_DECL, 395
MEMORYPOOL_DECL, 394
memory_heap, 544
header, 546
mtx, 546
provider, 546
memory_heap_t
Memory Heaps, 388
memory_pool_t, 546
align, 547
next, 547
object_size, 547
provider, 547
MEMORYPOOL_DECL
Memory Pools, 394
mpool
ch_thread, 522
MS2RTC
System Management, 109
msg
trace_event_t, 561
MSG_JOB_NULL
Jobs Queues, 364
MSG_OK
Scheduler, 87
MSG_RESET
Scheduler, 87

msg_t
OS Types and Structures, 64
MSG_TIMEOUT
Scheduler, 87
msgqueue
ch_thread, 521
mtx
ch_objects_factory, 490
memory_heap, 546
mtxlist
ch_thread, 521
MUTEX_DECL
Mutexes, 223
mutex_t
Mutexes, 224
Mutexes, 221
__MUTEX_DATA, 223
chMtxGetNextMutexX, 232
chMtxGetOwnerI, 232
chMtxLock, 224
chMtxLockS, 225
chMtxObjectInit, 224
chMtxQueueNotEmptyS, 231
chMtxTryLock, 226
chMtxTryLockS, 227
chMtxUnlock, 228
chMtxUnlockAll, 231
chMtxUnlockAllS, 230
chMtxUnlockS, 229
MUTEX_DECL, 223
mutex_t, 224

n
time_measurement_t, 555
n_ctxswc
kernel_stats_t, 540
n_irq
kernel_stats_t, 540
name
ch_dyn_element, 473
ch_os_instance_config, 503
ch_thread, 518
thread_descriptor_t, 553
trace_event_t, 562
next
ch_delta_list, 471
ch_dyn_element, 472
ch_dyn_list, 474
ch_list, 483
ch_mutex, 485
ch_priority_queue, 505
ch_queue, 506
event_listener, 534
event_source, 536
heap_header, 538
memory_pool_t, 547
pool_header, 551
NOPRIO
Scheduler, 87
NORMALPRIO
Scheduler, 88
NPAGES
Memory Heaps, 387
ntp
trace_event_t, 561

obj_flags
ch_oc_object, 499
obj_group
ch_oc_object, 498
obj_key
ch_oc_object, 498
obj_list
ch_objects_factory, 490
obj_pool
ch_objects_factory, 490
obj_sem
ch_oc_object, 498
object_size
memory_pool_t, 547
Objects Caches, 427
chCacheGetObject, 435
chCacheObjectInit, 434
chCacheReadObject, 437
chCacheReleaseObject, 438
chCacheReleaseObjectI, 436
chCacheWriteObject, 437
hash_get_s, 432
HASH_INSERT, 428
HASH_REMOVE, 429
lru_get_last_s, 433
LRU_INSERT_HEAD, 429
LRU_INSERT_TAIL, 429
LRU_REMOVE, 429
objects_cache_t, 431
OC_FLAG_FORGET, 430
OC_FLAG_INHASH, 430
OC_FLAG_INLRU, 430
OC_FLAG_LAZYWRITE, 430
OC_FLAG_NOTSYNC, 430
OC_FLAG_SHARED, 430
oc_flags_t, 430
OC_HASH_FUNCTION, 428
oc_hash_header_t, 430
oc_lru_header_t, 431
oc_object_t, 431
oc_readf_t, 431
oc_writef_t, 431
Objects FIFOs, 413
chFifoObjectInit, 415
chFifoObjectInitAligned, 414
chFifoReceiveObjectI, 424
chFifoReceiveObjectTimeout, 426
chFifoReceiveObjectTimeoutS, 425
chFifoReturnObject, 419
chFifoReturnObjectI, 418
chFifoReturnObjectS, 419
chFifoSendObject, 421

chFifoSendObjectAhead, 423
chFifoSendObjectAheadI, 422
chFifoSendObjectAheadS, 423
chFifoSendObjectI, 420
chFifoSendObjectS, 421
chFifoTakeObjectI, 416
chFifoTakeObjectTimeout, 417
chFifoTakeObjectTimeoutS, 416
objects_fifo_t, 414
objects_cache_t
Objects Caches, 431
objects_factory_t
Dynamic Objects Factory, 446
objects_fifo_t
Objects FIFOs, 414
objn
ch_objects_cache, 487
objp
ch_registered_static_object, 509
objsz
ch_objects_cache, 487
objvp
ch_objects_cache, 487
OC_FLAG_FORGET
Objects Caches, 430
OC_FLAG_INHASH
Objects Caches, 430
OC_FLAG_INLRU
Objects Caches, 430
OC_FLAG_LAZYWRITE
Objects Caches, 430
OC_FLAG_NOTSYNC
Objects Caches, 430
OC_FLAG_SHARED
Objects Caches, 430
oc_flags_t
Objects Caches, 430
OC_HASH_FUNCTION
Objects Caches, 428
oc_hash_header_t
Objects Caches, 430
oc_lru_header_t
Objects Caches, 431
oc_object_t
Objects Caches, 431
oc_readf_t
Objects Caches, 431
oc_writef_t
Objects Caches, 431
off_ctx
chdebug_t, 529
off_flags
chdebug_t, 530
off_inst_core_id
chdebug_t, 532
off_inst_reglist
chdebug_t, 532
off_inst_rfcu
chdebug_t, 532
off_inst_rlist
chdebug_t, 531
off_inst_rlist_current
chdebug_t, 531
off_inst_vtlist
chdebug_t, 532
off_name
chdebug_t, 529
off_newer
chdebug_t, 529
off_older
chdebug_t, 529
off_preempt
chdebug_t, 530
off_prio
chdebug_t, 529
off_refs
chdebug_t, 530
off_reserved
chdebug_t, 530
off_state
chdebug_t, 530
off_stklimit
chdebug_t, 529
off_sys_instances
chdebug_t, 531
off_sys_reglist
chdebug_t, 531
off_sys_reserved
chdebug_t, 531
off_sys_rfcu
chdebug_t, 531
off_sys_state
chdebug_t, 531
off_time
chdebug_t, 530
offset
tm_calibration_t, 556
Options, 36
_CHIBIOS_RT_CONF_, 39
_CHIBIOS_RT_CONF_VER_7_0_, 39
CH_CFG_CONTEXT_SWITCH_HOOK, 54
CH_CFG_FACTORY_GENERIC_BUFFERS, 48
CH_CFG_FACTORY_MAILBOXES, 48
CH_CFG_FACTORY_MAX_NAMES_LENGTH, 48
CH_CFG_FACTORY_OBJ_FIFOS, 49
CH_CFG_FACTORY_OBJECTS_REGISTRY, 48
CH_CFG_FACTORY_PIPES, 49
CH_CFG_FACTORY_SEMAPHORES, 48
CH_CFG_IDLE_ENTER_HOOK, 54
CH_CFG_IDLE_LEAVE_HOOK, 55
CH_CFG_IDLE_LOOP_HOOK, 55
CH_CFG_INTERVALS_SIZE, 40
CH_CFG_IRQ_EPILOGUE_HOOK, 54
CH_CFG_IRQ_PROLOGUE_HOOK, 54
CH_CFG_MEMCORE_SIZE, 46
CH_CFG_NO_IDLE_THREAD, 41

CH_CFG_OPTIMIZE_SPEED, 41
CH_CFG_OS_INSTANCE_EXTRA_FIELDS, 51
CH_CFG_OS_INSTANCE_INIT_HOOK, 52
CH_CFG_RUNTIME_FAULTS_HOOK, 56
CH_CFG_SMP_MODE, 39
CH_CFG_ST_FREQUENCY, 40
CH_CFG_ST_RESOLUTION, 40
CH_CFG_ST_TIMEDELTA, 40
CH_CFG_SYSTEM_EXTRA_FIELDS, 51
CH_CFG_SYSTEM_HALT_HOOK, 55
CH_CFG_SYSTEM_INIT_HOOK, 51
CH_CFG_SYSTEM_TICK_HOOK, 55
CH_CFG_THREAD_EXIT_HOOK, 53
CH_CFG_THREAD_EXTRA_FIELDS, 53
CH_CFG_THREAD_INIT_HOOK, 53
CH_CFG_TIME_QUANTUM, 41
CH_CFG_TIME_TYPES_SIZE, 40
CH_CFG_TRACE_HOOK, 56
CH_CFG_USE_CONDVARS, 43
CH_CFG_USE_CONDVARS_TIMEOUT, 44
CH_CFG_USE_DELEGATES, 47
CH_CFG_USE_DYNAMIC, 45
CH_CFG_USE_EVENTS, 44
CH_CFG_USE_EVENTS_TIMEOUT, 44
CH_CFG_USE_FACTORY, 48
CH_CFG_USE_HEAP, 46
CH_CFG_USE_JOBS, 47
CH_CFG_USE_MAILBOXES, 45
CH_CFG_USE_MEMCORE, 45
CH_CFG_USE_MEMPOOLS, 46
CH_CFG_USE_MESSAGES, 44
CH_CFG_USE_MESSAGES_PRIORITY, 45
CH_CFG_USE_MUTEXES, 43
CH_CFG_USE_MUTEXES_RECURSIVE, 43
CH_CFG_USE_OBJ_CACHES, 47
CH_CFG_USE_OBJ_FIFOS, 46
CH_CFG_USE_PIPES, 47
CH_CFG_USE_REGISTRY, 42
CH_CFG_USE_SEMAPHORES, 42
CH_CFG_USE_SEMAPHORES_PRIORITY, 43
CH_CFG_USE_TIMESTAMP, 42
CH_CFG_USE_TM, 41
CH_CFG_USE_WAITEXIT, 42
CH_DBG_ENABLE_ASSERTS, 49
CH_DBG_ENABLE_CHECKS, 49
CH_DBG_ENABLE_STACK_CHECK, 50
CH_DBG_FILL_THREADS, 50
CH_DBG_STATISTICS, 49
CH_DBG_SYSTEM_STATE_CHECK, 49
CH_DBG_THREADS_PROFILING, 51
CH_DBG_TRACE_BUFFER_SIZE, 50
CH_DBG_TRACE_MASK, 50
OS Instances, 67
__idle_thread, 68
__instance_get_currthread, 68
__instance_set_currthread, 68
chInstanceObjectInit, 69
OS Library, 314
OS Types and Structures, 59
__CH_OFFSETOF, 61
__CH_STRINGIFY, 60
__CH_USED, 61
ch_sys_halted, 67
ch_sys_initializing, 67
ch_sys_running, 67
ch_sys_uninit, 67
ch_system_t, 66
chSysHalt, 67
cnt_t, 64
core_id_t, 64
eventflags_t, 64
eventid_t, 64
eventmask_t, 64
likely, 61
msg_t, 64
os_instance_config_t, 66
os_instance_t, 65
ready_list_t, 66
registry_t, 66
rtcnt_t, 63
rttime_t, 63
stkalign_t, 63
syssts_t, 63
system_state_t, 66
thread_reference_t, 66
thread_t, 65
threadref, 62
threads_queue_t, 66
tmode_t, 63
tprio_t, 64
trefs_t, 63
tslices_t, 63
tstate_t, 63
ucnt_t, 64
unlikely, 62
virtual_timer_t, 65
virtual_timers_list_t, 65
vtfunc_t, 65
os_instance_config_t
OS Types and Structures, 66
os_instance_t
OS Types and Structures, 65
owner
ch_mutex, 485
ch_thread, 518

pages
heap_header, 538
panic_msg
ch_system_debug, 515
par
ch_virtual_timer, 524
PC_INIT
Pipes, 345
PC_LOCK
Pipes, 345
PC_UNLOCK

Pipes, 345
pipe
ch_dyn_pipe, 479
PIPE_DECL
Pipes, 347
pipe_list
ch_objects_factory, 491
pipe_read
Pipes, 348
pipe_t, 548
buffer, 549
cmtx, 550
cnt, 549
rdptr, 549
reset, 550
rmtx, 550
rtr, 550
top, 549
wmtx, 550
wrptr, 549
wtr, 550
pipe_write
Pipes, 348
Pipes, 344
__PIPE_DATA, 346
chPipeGetFreeCount, 352
chPipeGetSize, 352
chPipeGetUsedCount, 352
chPipeObjectInit, 349
chPipeReadTimeout, 351
chPipeReset, 349
chPipeResume, 353
chPipeWriteTimeout, 350
PC_INIT, 345
PC_LOCK, 345
PC_UNLOCK, 345
PIPE_DECL, 347
pipe_read, 348
pipe_write, 348
PR_INIT, 346
PR_LOCK, 346
PR_UNLOCK, 346
PW_INIT, 345
PW_LOCK, 346
PW_UNLOCK, 346
pool
guarded_memory_pool_t, 537
pool_header, 551
next, 551
Port Interface, 58
CH_PORT_SUPPORTS_RECURSIVE_LOCKS, 58
PORT_CORES_NUMBER, 58
PORT_CORES_NUMBER
Port Interface, 58
pqueue
ch_ready_list, 507
ch_thread, 517
PR_INIT
Pipes, 346
PR_LOCK
Pipes, 346
PR_UNLOCK
Pipes, 346
prev
ch_delta_list, 471
ch_priority_queue, 505
ch_queue, 506
prio
ch_priority_queue, 505
thread_descriptor_t, 553
provider
memory_heap, 546
memory_pool_t, 547
ptr
trace_buffer_t, 558
ptrsize
chdebug_t, 528
PW_INIT
Pipes, 345
PW_LOCK
Pipes, 346
PW_UNLOCK
Pipes, 346

qr
mailbox_t, 543
queue
ch_mutex, 485
ch_registry, 510
ch_semaphore, 512
ch_thread, 517
ch_threads_queue, 523
condition_variable, 533
qw
mailbox_t, 542

rdptr
mailbox_t, 542
pipe_t, 549
rdy
trace_event_t, 561
rdymsg
ch_thread, 519
readf
ch_objects_cache, 488
ready_list_t
OS Types and Structures, 66
realprio
ch_thread, 521
reason
trace_event_t, 562
refs
ch_dyn_element, 472
ch_thread, 518
REG_HEADER
Registry, 281

REG_INSERT
Registry, 282
REG_REMOVE
Registry, 281
registered_object_t
Dynamic Objects Factory, 445
Registry, 280
__reg_object_init, 286
ch_debug, 288
chRegFindThreadByName, 284
chRegFindThreadByPointer, 285
chRegFindThreadByWorkingArea, 285
chRegFirstThread, 282
chRegGetThreadNameX, 287
chRegNextThread, 283
chRegSetThreadName, 287
chRegSetThreadNameX, 288
REG_HEADER, 281
REG_INSERT, 282
REG_REMOVE, 281
registry_t
OS Types and Structures, 66
reglist
ch_os_instance, 501
ch_system, 513
Release and Licensing, 25
Release Information, 25
__CHIBIOS__, 25
CH_VERSION, 26
CH_VERSION_DATE, 26
CH_VERSION_MONTH, 26
CH_VERSION_NICKNAME, 26
CH_VERSION_PATCH, 26
CH_VERSION_STABLE, 25
CH_VERSION_YEAR, 26
reload
ch_virtual_timer, 524
reset
mailbox_t, 542
pipe_t, 550
Restrictions, 56
CH_CFG_ST_TIMEDELTA, 57
CH_CFG_USE_CONDVARS, 57
CH_CFG_USE_DYNAMIC, 57
CH_CFG_USE_MUTEXES, 57
CH_CFG_USE_TM, 57
CH_DBG_STATISTICS, 57
CH_DBG_TRACE_MASK, 57
rfcu
ch_os_instance, 501
ch_system, 513
rfcu_mask_t
Runtime Faults Collection Unit, 71
rfcu_t
Runtime Faults Collection Unit, 71
rlist
ch_os_instance, 501
rmtx
pipe_t, 550
rqueue
ch_thread, 518
RT Kernel, 33
rt.dox, 592
rt_arch.dox, 592
RTC2MS
System Management, 111
RTC2S
System Management, 110
RTC2US
System Management, 111
rtcnt_t
OS Types and Structures, 63
rtr
pipe_t, 550
rtstamp
trace_event_t, 560
rttime_t
OS Types and Structures, 63
Runtime Faults Collection Unit, 70
__rfcu_object_init, 72
CH_RFCU_ALL_FAULTS, 70
CH_RFCU_VT_INSUFFICIENT_DELTA, 70
CH_RFCU_VT_SKIPPED_DEADLINE, 70
chRFCUCollectFaultsI, 71
chRFCUGetAndClearFaultsI, 71
rfcu_mask_t, 71
rfcu_t, 71

S2RTC
System Management, 108
Scheduler, 84
__sch_get_currthread, 91
__sch_ready_ahead, 92
__sch_ready_behind, 91
__sch_reschedule_ahead, 93
__sch_reschedule_behind, 93
__sch_wakeup, 94
CH_FLAG_MODE_HEAP, 90
CH_FLAG_MODE_MASK, 90
CH_FLAG_MODE_MPOOL, 90
CH_FLAG_MODE_STATIC, 90
CH_FLAG_TERMINATE, 91
ch_sch_prio_insert, 94
CH_STATE_CURRENT, 88
CH_STATE_FINAL, 90
CH_STATE_NAMES, 90
CH_STATE_QUEUED, 88
CH_STATE_READY, 88
CH_STATE_SLEEPING, 89
CH_STATE_SNDMSG, 89
CH_STATE_SNDMSGQ, 89
CH_STATE_SUSPENDED, 88
CH_STATE_WTANDEVT, 89
CH_STATE_WTCOND, 89
CH_STATE_WTEXIT, 89
CH_STATE_WTMSG, 90
CH_STATE_WTMTX, 89

CH_STATE_WTOREVT, 89
CH_STATE_WTSEM, 88
CH_STATE_WTSTART, 88
chSchDoPreemption, 100
chSchDoYieldS, 101
chSchGoSleepS, 96
chSchGoSleepTimeoutS, 96
chSchIsPreemptionRequired, 100
chSchObjectInit, 102
chSchPreemption, 101
chSchReadyI, 95
chSchRescheduleS, 99
chSchSelectFirst, 102
chSchWakeupS, 98
firstprio, 91
HIGHPRIO, 88
IDLEPRIO, 87
LOWPRIO, 87
MSG_OK, 87
MSG_RESET, 87
MSG_TIMEOUT, 87
NOPRIO, 87
NORMALPRIO, 88
sem
ch_binary_semaphore, 470
ch_dyn_semaphore, 480
guarded_memory_pool_t, 537
sem_insert
Counting Semaphores, 208
sem_list
ch_objects_factory, 490
sem_pool
ch_objects_factory, 490
SEMAPHORE_DECL
Counting Semaphores, 208
semaphore_t
Counting Semaphores, 209
sentmsg
ch_thread, 520
size
chdebug_t, 528
heap_header, 538
trace_buffer_t, 558
state
ch_system, 513
ch_thread, 518
trace_event_t, 560
Statistics, 310
__stats_ctxswc, 311
__stats_increase_irq, 311
__stats_init, 313
__stats_object_init, 313
__stats_start_measure_crit_isr, 312
__stats_start_measure_crit_thd, 311
__stats_stop_measure_crit_isr, 312
__stats_stop_measure_crit_thd, 312
stats
ch_thread, 522
stkalign_t
OS Types and Structures, 63
suspended
trace_buffer_t, 558
sw
trace_event_t, 561
Synchronization, 206, 317
Synchronous Messages, 266
__ch_msg_insert, 268
chMsgGet, 276
chMsgIsPendingI, 275
chMsgPoll, 274
chMsgPollS, 270
chMsgRelease, 271
chMsgReleaseS, 276
chMsgSend, 268
chMsgWait, 272
chMsgWaitS, 268
chMsgWaitTimeout, 273
chMsgWaitTimeoutS, 269
sysinterval_t
Time and Intervals, 133
syssts_t
OS Types and Structures, 63
System, 58
System Management, 103
__c1_main_thread_stack_base__, 125
__c1_main_thread_stack_end__, 125
__main_thread_stack_base__, 125
__main_thread_stack_end__, 125
ch0, 124
ch1, 125
ch_core0_cfg, 125
ch_core1_cfg, 125
CH_FAST_IRQ_HANDLER, 108
CH_INTEGRITY_PORT, 106
CH_INTEGRITY_REGISTRY, 106
CH_INTEGRITY_RLIST, 106
CH_INTEGRITY_VTLIST, 106
CH_IRQ_EPILOGUE, 107
CH_IRQ_HANDLER, 108
CH_IRQ_IS_VALID_KERNEL_PRIORITY, 107
CH_IRQ_IS_VALID_PRIORITY, 106
CH_IRQ_PROLOGUE, 107
CH_SYS_CORE0_MEMORY, 106
CH_SYS_CORE1_MEMORY, 106
ch_system, 124
chSysDisable, 120
chSysEnable, 120
chSysGetIdleThreadX, 115
chSysGetRealtimeCounterX, 112
chSysGetStatusAndLockX, 117
chSysHalt, 114
chSysInit, 113
chSysIntegrityCheckI, 115
chSysIsCounterWithinX, 118
chSysLock, 121
chSysLockFromISR, 122

chSysNotifyInstance, 124
chSysPolledDelayX, 119
chSysRestoreStatusX, 118
chSysSuspend, 120
chSysSwitch, 112
chSysTimerHandlerI, 116
chSysUnconditionalLock, 123
chSysUnconditionalUnlock, 123
chSysUnlock, 121
chSysUnlockFromISR, 122
chSysWaitSystemState, 113
currcore, 106
MS2RTC, 109
RTC2MS, 111
RTC2S, 110
RTC2US, 111
S2RTC, 108
THD_WORKING_AREA, 113
US2RTC, 110
system_debug_t
Checks and Assertions, 293
system_state_t
OS Types and Structures, 66
systime
ch_virtual_timers_list, 525
systime_t
Time and Intervals, 133
systimestamp_t
Time and Intervals, 134

tfunc_t
Threads, 174
THD_DESCRIPTOR
Threads, 171
THD_DESCRIPTOR_AFFINITY
Threads, 171
THD_FUNCTION
Threads, 171
THD_WORKING_AREA
System Management, 113
Threads, 170
THD_WORKING_AREA_BASE
Threads, 170
THD_WORKING_AREA_END
Threads, 170
THD_WORKING_AREA_SIZE
Threads, 169
thread_descriptor_t, 551
arg, 553
funcp, 553
instance, 553
name, 553
prio, 553
wbase, 553
wend, 553
thread_reference_t
OS Types and Structures, 66
thread_t
OS Types and Structures, 65
threadref
OS Types and Structures, 62
Threads, 166
__THREADS_QUEUE_DATA, 169
__thd_object_init, 174
__thd_stackfill, 174
chThdAddRef, 181
chThdCreate, 178
chThdCreateI, 177
chThdCreateStatic, 179
chThdCreateSuspended, 176
chThdCreateSuspendedI, 175
chThdDequeueAllI, 196
chThdDequeueNextI, 195
chThdDoDequeueNextI, 201
chThdEnqueueTimeoutS, 194
chThdExit, 183
chThdExitS, 184
chThdGetPriorityX, 197
chThdGetSelfX, 196
chThdGetTicksX, 197
chThdGetWorkingAreaX, 198
chThdQueueIsEmptyI, 201
chThdQueueObjectInit, 200
chThdRelease, 182
chThdResume, 194
chThdResumeI, 192
chThdResumeS, 193
chThdSetPriority, 186
chThdShouldTerminateX, 198
chThdSleep, 187
chThdSleepMicroseconds, 173
chThdSleepMilliseconds, 172
chThdSleepS, 199
chThdSleepSeconds, 172
chThdSleepUntil, 188
chThdSleepUntilWindowed, 189
chThdStart, 180
chThdStartI, 199
chThdSuspendS, 190
chThdSuspendTimeoutS, 191
chThdTerminate, 187
chThdTerminatedX, 198
chThdWait, 185
chThdYield, 190
tfunc_t, 174
THD_DESCRIPTOR, 171
THD_DESCRIPTOR_AFFINITY, 171
THD_FUNCTION, 171
THD_WORKING_AREA, 170
THD_WORKING_AREA_BASE, 170
THD_WORKING_AREA_END, 170
THD_WORKING_AREA_SIZE, 169
THREADS_QUEUE_DECL, 169
THREADS_QUEUE_DECL
Threads, 169
threads_queue_t
OS Types and Structures, 66

threadsize
chdebug_t, 529
ticks
ch_thread, 519
time
ch_thread, 519
trace_event_t, 561
Time and Intervals, 128
chTimeAddX, 138
chTimeDiffX, 138
chTimeI2MS, 137
chTimeI2S, 136
chTimeI2US, 137
chTimeIsInRangeX, 138
chTimeMS2I, 135
chTimeS2I, 135
chTimeStampAddX, 139
chTimeStampDiffX, 139
chTimeStampIsInRangeX, 140
chTimeUS2I, 136
sysinterval_t, 133
systime_t, 133
systimestamp_t, 134
time_conv_t, 134
TIME_I2MS, 132
TIME_I2S, 132
TIME_I2US, 133
TIME_IMMEDIATE, 129
TIME_INFINITE, 129
TIME_MAX_INTERVAL, 130
TIME_MAX_SYSTIME, 130
TIME_MS2I, 130
time_msecs_t, 134
TIME_S2I, 130
time_secs_t, 134
TIME_US2I, 131
time_usecs_t, 134
Time Measurement, 202
__tm_calibration_object_init, 205
chTMChainMeasurementToX, 204
chTMObjectInit, 203
chTMStartMeasurementX, 204
chTMStopMeasurementX, 204
TM_CALIBRATION_LOOP, 203
tm_stop, 203
time_conv_t
Time and Intervals, 134
TIME_I2MS
Time and Intervals, 132
TIME_I2S
Time and Intervals, 132
TIME_I2US
Time and Intervals, 133
TIME_IMMEDIATE
Time and Intervals, 129
TIME_INFINITE
Time and Intervals, 129
TIME_MAX_INTERVAL
Time and Intervals, 130
TIME_MAX_SYSTIME
Time and Intervals, 130
time_measurement_t, 554
best, 555
cumulative, 555
last, 555
n, 555
worst, 555
TIME_MS2I
Time and Intervals, 130
time_msecs_t
Time and Intervals, 134
TIME_S2I
Time and Intervals, 130
time_secs_t
Time and Intervals, 134
TIME_US2I
Time and Intervals, 131
time_usecs_t
Time and Intervals, 134
timesize
chdebug_t, 529
TM_CALIBRATION_LOOP
Time Measurement, 203
tm_calibration_t, 555
offset, 556
tm_stop
Time Measurement, 203
tmc
ch_system, 513
tmode_t
OS Types and Structures, 63
top
mailbox_t, 542
pipe_t, 549
topmem
memcore_t, 543
tp
trace_event_t, 561
tprio_t
OS Types and Structures, 64
trace_buffer
ch_os_instance, 502
trace_buffer_t, 556
buffer, 558
ptr, 558
size, 558
suspended, 558
trace_event_t, 559
halt, 562
isr, 562
msg, 561
name, 562
ntp, 561
rdy, 561
reason, 562
rtstamp, 560

state, 560
sw, 561
time, 561
tp, 561
type, 560
u, 562
up1, 562
up2, 562
user, 562
wtobjp, 561
trace_next
Tracing, 304
Tracing, 299
__trace_halt, 304, 307
__trace_isr_enter, 303, 306
__trace_isr_leave, 304, 306
__trace_object_init, 304
__trace_ready, 303, 305
__trace_switch, 303, 305
CH_DBG_TRACE_BUFFER_SIZE, 303
CH_DBG_TRACE_MASK, 303
CH_DBG_TRACE_MASK_ALL, 303
CH_DBG_TRACE_MASK_DISABLED, 302
CH_DBG_TRACE_MASK_HALT, 302
CH_DBG_TRACE_MASK_ISR, 302
CH_DBG_TRACE_MASK_NONE, 302
CH_DBG_TRACE_MASK_READY, 302
CH_DBG_TRACE_MASK_SLOW, 302
CH_DBG_TRACE_MASK_SWITCH, 302
CH_DBG_TRACE_MASK_USER, 302
CH_TRACE_TYPE_HALT, 302
CH_TRACE_TYPE_ISR_ENTER, 301
CH_TRACE_TYPE_ISR_LEAVE, 301
CH_TRACE_TYPE_READY, 301
CH_TRACE_TYPE_SWITCH, 301
CH_TRACE_TYPE_UNUSED, 301
CH_TRACE_TYPE_USER, 302
chDbgWriteTrace, 304
chDbgWriteTraceI, 304
chTraceResume, 310
chTraceResumeI, 309
chTraceSuspend, 309
chTraceSuspendI, 308
chTraceWrite, 308
chTraceWriteI, 307
trace_next, 304
trefs_t
OS Types and Structures, 63
TRUE
Version Numbers and Identification, 35
tslices_t
OS Types and Structures, 63
tstate_t
OS Types and Structures, 63
type
trace_event_t, 560

u
ch_thread, 521
trace_event_t, 562
ucnt_t
OS Types and Structures, 64
unlikely
OS Types and Structures, 62
up1
trace_event_t, 562
up2
trace_event_t, 562
US2RTC
System Management, 110
used
heap_header, 539
user
trace_event_t, 562

version
chdebug_t, 528
Version Numbers and Identification, 34, 314
__CHIBIOS_OSLIB__, 315
__CHIBIOS_RT__, 34
__oslib_init, 317
CH_CFG_USE_DELEGATES, 316
CH_CFG_USE_FACTORY, 315
CH_CFG_USE_HEAP, 316
CH_CFG_USE_JOBS, 316
CH_CFG_USE_MAILBOXES, 316
CH_CFG_USE_MEMPOOLS, 316
CH_CFG_USE_OBJ_CACHES, 316
CH_CFG_USE_OBJ_FIFOS, 316
CH_CFG_USE_PIPES, 316
CH_KERNEL_MAJOR, 35
CH_KERNEL_MINOR, 35
CH_KERNEL_PATCH, 35
CH_KERNEL_STABLE, 34
CH_KERNEL_VERSION, 34
CH_OSLIB_MAJOR, 315
CH_OSLIB_MINOR, 315
CH_OSLIB_PATCH, 315
CH_OSLIB_STABLE, 315
CH_OSLIB_VERSION, 315
FALSE, 35
TRUE, 35
Virtual Timers, 141
__vt_object_init, 165
chVTDoResetI, 147
chVTDoSetContinuousI, 146
chVTDoSetI, 145
chVTDoTickI, 149
chVTGetReloadIntervalX, 163
chVTGetRemainingIntervalI, 148
chVTGetSystemTime, 152
chVTGetSystemTimeX, 152
chVTGetTimersStateI, 155
chVTGetTimeStamp, 164
chVTGetTimeStampI, 150
chVTIsArmed, 157
chVTIsArmedI, 156
chVTIsSystemTimeWithin, 155

chVTIsSystemTimeWithinX, 154
chVTObjectInit, 151
chVTReset, 158
chVTResetI, 157
chVTResetTimeStamp, 164
chVTResetTimeStampI, 151
chVTSet, 160
chVTSetContinuous, 162
chVTSetContinuousI, 161
chVTSetI, 159
chVTSetReloadIntervalX, 163
chVTTimeElapsedSinceX, 153
vt_enqueue, 144
vt_insert_first, 143
vt_set_alarm, 142
virtual_timer_t
OS Types and Structures, 65
virtual_timers_list_t
OS Types and Structures, 65
vt_enqueue
Virtual Timers, 144
vt_insert_first
Virtual Timers, 143
vt_set_alarm
Virtual Timers, 142
vtfunc_t
OS Types and Structures, 65
vtlist
ch_os_instance, 501

wabase
ch_thread, 518
waiting
ch_thread, 521
wbase
thread_descriptor_t, 553
wend
thread_descriptor_t, 553
wflags
event_listener, 534
wmtx
pipe_t, 550
worst
time_measurement_t, 555
writef
ch_objects_cache, 488
wrptr
mailbox_t, 542
pipe_t, 549
wtmtxp
ch_thread, 520
wtobjp
ch_thread, 519
trace_event_t, 561
wtr
pipe_t, 550
wtsemp
ch_thread, 520
wttrp
ch_thread, 520

zero
chdebug_t, 528