ChibiOS/NIL

3.2.2

Reference Manual

Thu Apr 30 2020 11:07:20

Contents

Chapter 1

ChibiOS/NIL

1.1 Copyright

1.2 Introduction

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

1.3 Related Documents

- ChibiOS/NIL General Architecture

Chapter 2

Module Index

2.1 Modules

Here is a list of all modules:

Chapter 3

Hierarchical Index

3.1 Class Hierarchy

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

Chapter 4

Data Structure Index

4.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 5

File Index

5.1 File List

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

Chapter 6

Module Documentation

6.1 NIL Kernel

6.1.1 Detailed Description

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

Modules

- Configuration
- API

6.2 Configuration

6.2.1 Detailed Description

Kernel related settings and hooks.

Kernel parameters and options

- #define CH_CFG_NUM_THREADS 3

  Number of user threads in the application.

System timer settings

- #define CH_CFG_ST_RESOLUTION 32

  System time counter resolution.
- #define CH_CFG_ST_FREQUENCY 1000

  System tick frequency.
- #define CH_CFG_ST_TIMEDELTA 0

  Time delta constant for the tick-less mode.

Subsystem options

- #define CH_CFG_USE_SEMAPHORES TRUE

  Semaphores APIs.
- #define CH_CFG_USE_MUTEXES FALSE

  Mutexes APIs.
- #define CH_CFG_USE_EVENTS TRUE

  Events Flags APIs.
- #define CH_CFG_USE_MAILBOXES TRUE

  Mailboxes APIs.
- #define CH_CFG_USE_MEMCORE TRUE

  Core Memory Manager APIs.
- #define CH_CFG_USE_HEAP TRUE

  Heap Allocator APIs.
- #define CH_CFG_USE_MEMPOOLS TRUE

  Memory Pools Allocator APIs.
- #define CH_CFG_USE_OBJ_FIFOS TRUE

  Objects FIFOs APIs.
- #define CH_CFG_USE_PIPES TRUE

  Pipes APIs.
- #define CH_CFG_MEMCORE_SIZE 0

  Managed RAM size.

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

  System assertions.
- #define CH_DBG_ENABLE_STACK_CHECK TRUE

  Stack check.

Kernel hooks

- #define CH_CFG_SYSTEM_INIT_HOOK()

  System initialization hook.
- #define CH_CFG_THREAD_EXT_FIELDS /∗ Add threads custom fields here.∗/

  Threads descriptor structure extension.
- #define CH_CFG_THREAD_EXT_INIT_HOOK(tr)

  Threads initialization hook.
- #define CH_CFG_IDLE_ENTER_HOOK()

  Idle thread enter hook.
- #define CH_CFG_IDLE_LEAVE_HOOK()

  Idle thread leave hook.
- #define CH_CFG_SYSTEM_HALT_HOOK(reason)

  System halt hook.

6.2.2 Macro Definition Documentation

6.2.2.1 #define CH_CFG_NUM_THREADS 3

Number of user threads in the application.

Note

This number is not inclusive of the idle thread which is Implicitly handled.

6.2.2.2 #define CH_CFG_ST_RESOLUTION 32

System time counter resolution.

Note

Allowed values are 16 or 32 bits.

6.2.2.3 #define CH_CFG_ST_FREQUENCY 1000

System tick frequency.

Note

This value together with the CH_CFG_ST_RESOLUTION option defines the maximum amount of time allowed for timeouts.

6.2.2.4 #define CH_CFG_ST_TIMEDELTA 0

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.

6.2.2.5 #define CH_CFG_USE_SEMAPHORES TRUE

Semaphores APIs.

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

Note

The default is TRUE.

6.2.2.6 #define CH_CFG_USE_MUTEXES FALSE

Mutexes APIs.

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

Note

Feature not currently implemented.
The default is FALSE.

6.2.2.7 #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.

6.2.2.8 #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`.

6.2.2.9 #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`.

6.2.2.10 #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`.

6.2.2.11 #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`.

6.2.2.12 #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`.

6.2.2.13 #define CH_CFG_USE_PIPES TRUE

Pipes APIs.

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

Note

The default is `TRUE`.

6.2.2.14 #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.

6.2.2.15 #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.

6.2.2.16 #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.

6.2.2.17 #define CH_CFG_FACTORY_OBJECTS_REGISTRY TRUE

Enables the registry of generic objects.

6.2.2.18 #define CH_CFG_FACTORY_GENERIC_BUFFERS TRUE

Enables factory for generic buffers.

6.2.2.19 #define CH_CFG_FACTORY_SEMAPHORES TRUE

Enables factory for semaphores.

6.2.2.20 #define CH_CFG_FACTORY_MAILBOXES TRUE

Enables factory for mailboxes.

6.2.2.21 #define CH_CFG_FACTORY_OBJ_FIFOS TRUE

Enables factory for objects FIFOs.

6.2.2.22 #define CH_CFG_FACTORY_PIPES TRUE

Enables factory for Pipes.

6.2.2.23 #define CH_DBG_STATISTICS FALSE

Debug option, kernel statistics.

Note

Feature not currently implemented.
The default is `FALSE`.

6.2.2.24 #define CH_DBG_SYSTEM_STATE_CHECK TRUE

Debug option, system state check.

Note

The default is `FALSE`.

6.2.2.25 #define CH_DBG_ENABLE_CHECKS TRUE

Debug option, parameters checks.

Note

The default is `FALSE`.

6.2.2.26 #define CH_DBG_ENABLE_ASSERTS TRUE

System assertions.

Note

The default is `FALSE`.

6.2.2.27 #define CH_DBG_ENABLE_STACK_CHECK TRUE

Stack check.

Note

The default is `FALSE`.

6.2.2.28 #define CH_CFG_SYSTEM_INIT_HOOK( )

Value:

```
{                                       \
}
```

System initialization hook.

6.2.2.29 #define CH_CFG_THREAD_EXT_FIELDS /* Add threads custom fields here.*/

Threads descriptor structure extension.

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

6.2.2.30 #define CH_CFG_THREAD_EXT_INIT_HOOK(*tr*)

Value:

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

Threads initialization hook.

6.2.2.31 #define CH_CFG_IDLE_ENTER_HOOK( )

Value:

```
{                                               \
}
```

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.

6.2.2.32 #define CH_CFG_IDLE_LEAVE_HOOK( )

Value:

```
{                                               \
}
```

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.

6.2.2.33 #define CH_CFG_SYSTEM_HALT_HOOK(*reason*)

Value:

```
{                                       \
}
```

System halt hook.

6.3 API

6.3.1 Detailed Description

Macros

- #define _CHIBIOS_NIL_

  ChibiOS/NIL identification macro.
- #define CH_KERNEL_STABLE 1

  Stable release flag.
- #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 THD_IDLE_BASE (&__main_thread_stack_base__)
- #define __CH_STRINGIFY(a) #a

  Utility to make the parameter a quoted string.

ChibiOS/NIL version identification

- #define CH_KERNEL_VERSION "3.2.2"

  Kernel version string.
- #define CH_KERNEL_MAJOR 3

  Kernel version major number.
- #define CH_KERNEL_MINOR 2

  Kernel version minor number.
- #define CH_KERNEL_PATCH 2

  Kernel version patch number.

Constants for configuration options

- #define FALSE 0

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

  Generic 'true' preprocessor boolean constant.

Wakeup messages

- #define MSG_OK (msg_t)0
  OK wakeup message.
- #define MSG_TIMEOUT (msg_t)-1
  Wake-up caused by a timeout condition.
- #define MSG_RESET (msg_t)-2
  Wake-up caused by a reset condition.

Special time constants

- #define TIME_IMMEDIATE ((sysinterval_t)-1)
  Zero time specification for some functions with a timeout specification.
- #define TIME_INFINITE ((sysinterval_t)0)
  Infinite time 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.

Thread state related macros

- #define NIL_STATE_READY (tstate_t)0
  Thread ready or executing.
- #define NIL_STATE_SLEEPING (tstate_t)1
  Thread sleeping.
- #define NIL_STATE_SUSP (tstate_t)2
  Thread suspended.
- #define NIL_STATE_WTQUEUE (tstate_t)3
  On queue or semaph.
- #define NIL_STATE_WTOREVT (tstate_t)4
  Waiting for events.
- #define **NIL_THD_IS_READY**(tp) ((tp)->state == NIL_STATE_READY)
- #define **NIL_THD_IS_SLEEPING**(tp) ((tp)->state == NIL_STATE_SLEEPING)
- #define **NIL_THD_IS_SUSP**(tp) ((tp)->state == NIL_STATE_SUSP)
- #define **NIL_THD_IS_WTQUEUE**(tp) ((tp)->state == NIL_STATE_WTQUEUE)
- #define **NIL_THD_IS_WTOREVT**(tp) ((tp)->state == NIL_STATE_WTOREVT)

Events related macros

- #define ALL_EVENTS ((eventmask_t)-1)
  All events allowed mask.
- #define EVENT_MASK(eid) ((eventmask_t)(1 << (eid)))
  Returns an event mask from an event identifier.

Threads tables definition macros

- #define THD_TABLE_BEGIN const thread_config_t nil_thd_configs[CH_CFG_NUM_THREADS + 1] = {
  Start of user threads table.
- #define THD_TABLE_ENTRY(wap, name, funcp, arg)
  Entry of user threads table.
- #define THD_TABLE_END
  End of user threads table.

Memory alignment support macros

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

  Alignment mask constant.
- #define MEM_ALIGN_PREV(p, a) ((size_t)(p) & ~MEM_ALIGN_MASK(a))

  Aligns to the previous aligned memory address.
- #define MEM_ALIGN_NEXT(p, a)

  Aligns to the new aligned memory address.
- #define MEM_IS_ALIGNED(p, a) (((size_t)(p) & MEM_ALIGN_MASK(a)) == 0U)

  Returns whatever a pointer or memory size is aligned.
- #define MEM_IS_VALID_ALIGNMENT(a) (((size_t)(a) != 0U) && (((size_t)(a) & ((size_t)(a) - 1U)) == 0U))

  Returns whatever a constant is a valid alignment.

Working Areas

- #define THD_WORKING_AREA_SIZE(n)

  Calculates the total Working Area size.
- #define THD_WORKING_AREA(s, n) PORT_WORKING_AREA(s, n)

  Static working area allocation.

Threads abstraction macros

- #define THD_FUNCTION(tname, arg) PORT_THD_FUNCTION(tname, arg)

  Thread declaration macro.

ISRs abstraction macros

- #define CH_IRQ_IS_VALID_PRIORITY(prio) PORT_IRQ_IS_VALID_PRIORITY(prio)

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

  Priority level validation macro.
- #define CH_IRQ_PROLOGUE()

  IRQ handler enter code.
- #define CH_IRQ_EPILOGUE()

  IRQ handler exit code.
- #define CH_IRQ_HANDLER(id) PORT_IRQ_HANDLER(id)

  Standard normal IRQ handler declaration.

Fast ISRs abstraction macros

- #define CH_FAST_IRQ_HANDLER(id) PORT_FAST_IRQ_HANDLER(id)

  Standard fast IRQ handler declaration.

Time conversion utilities

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

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

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

Microseconds to time interval.

- #define TIME_I2S(interval)

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

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

  Time interval to microseconds.

Threads queues

- #define _THREADS_QUEUE_DATA(name) {(cnt_t)0}

  Data part of a static threads queue object initializer.
- #define _THREADS_QUEUE_DECL(name) threads_queue_t name = _THREADS_QUEUE_DATA(name)

  Static threads queue object initializer.

Semaphores macros

- #define _SEMAPHORE_DATA(name, n) {n}

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

  Static semaphore initializer.

Macro Functions

- #define chSysGetRealtimeCounterX() (rtcnt_t)port_rt_get_counter_value()

  Returns the current value of the system real time counter.
- #define chSysDisable()

  Raises the system interrupt priority mask to the maximum level.
- #define chSysSuspend()

  Raises the system interrupt priority mask to system level.
- #define chSysEnable()

  Lowers the system interrupt priority mask to user level.
- #define chSysLock()

  Enters the kernel lock state.
- #define chSysUnlock()

  Leaves the kernel lock state.
- #define chSysLockFromISR()

  Enters the kernel lock state from within an interrupt handler.
- #define chSysUnlockFromISR()

  Leaves the kernel lock state from within an interrupt handler.
- #define chSchIsRescRequiredI() ((bool)(nil.current != nil.next))

  Evaluates if a reschedule is required.
- #define chThdGetSelfX() nil.current

  Returns a pointer to the current `thread_t`.
- #define chThdSleepSeconds(secs) chThdSleep(TIME_S2I(secs))

  Delays the invoking thread for the specified number of seconds.
- #define chThdSleepMilliseconds(msecs) chThdSleep(TIME_MS2I(msecs))

  Delays the invoking thread for the specified number of milliseconds.
- #define chThdSleepMicroseconds(usecs) chThdSleep(TIME_US2I(usecs))

  Delays the invoking thread for the specified number of microseconds.

- #define chThdSleepS(timeout) (void) chSchGoSleepTimeoutS(NIL_STATE_SLEEPING, timeout)

  Suspends the invoking thread for the specified time.
- #define chThdSleepUntilS(abstime)

  Suspends the invoking thread until the system time arrives to the specified value.
- #define chThdQueueObjectInit(tqp) ((tqp)->cnt = (cnt_t)0)

  Initializes a threads queue object.
- #define chThdQueueIsEmptyI(tqp) ((bool)(tqp->cnt >= (cnt_t)0))

  Evaluates to `true` if the specified queue is empty.
- #define chSemObjectInit(sp, n) ((sp)->cnt = (n))

  Initializes a semaphore with the specified counter value.
- #define chSemWait(sp) chSemWaitTimeout(sp, TIME_INFINITE)

  Performs a wait operation on a semaphore.
- #define chSemWaitS(sp) chSemWaitTimeoutS(sp, TIME_INFINITE)

  Performs a wait operation on a semaphore.
- #define chSemFastWaitI(sp) ((sp)->cnt--)

  Decreases the semaphore counter.
- #define chSemFastSignalI(sp) ((sp)->cnt++)

  Increases the semaphore counter.
- #define chSemGetCounterI(sp) ((sp)->cnt)

  Returns the semaphore counter current value.
- #define chVTGetSystemTimeX() (nil.systime)

  Current system time.
- #define chVTTimeElapsedSinceX(start) chTimeDiffX((start), chVTGetSystemTimeX())

  Returns the elapsed time since the specified start time.
- #define chTimeAddX(systime, interval) ((systime_t)(systime) + (systime_t)(interval))

  Adds an interval to a system time returning a system time.
- #define chTimeDiffX(start, end) ((sysinterval_t)((systime_t)((systime_t)(end) - (systime_t)(start))))

  Subtracts two system times returning an interval.
- #define chTimeIsInRangeX(time, start, end)

  Checks if the specified time is within the specified time range.
- #define chDbgCheck(c)

  Function parameters check.
- #define chDbgAssert(c, r)

  Condition assertion.

Typedefs

- typedef uint32_t systime_t

  Type of system time.
- typedef uint32_t sysinterval_t

  Type of time interval.
- typedef uint64_t time_conv_t

  Type of time conversion variable.
- typedef struct nil_thread thread_t

  Type of a structure representing a thread.
- typedef struct nil_threads_queue threads_queue_t

  Type of a queue of threads.
- typedef threads_queue_t semaphore_t

  Type of a structure representing a semaphore.
- typedef void(* tfunc_t) (void *p)

Thread function.

- typedef struct nil_thread_cfg thread_config_t

  Type of a structure representing a thread static configuration.
- typedef thread_t * thread_reference_t

  Type of a thread reference.
- typedef struct nil_system nil_system_t

  Type of a structure representing the system.

Data Structures

- struct nil_threads_queue

  Structure representing a queue of threads.
- struct nil_thread_cfg

  Structure representing a thread static configuration.
- struct nil_thread

  Structure representing a thread.
- struct nil_system

  System data structure.

Functions

- static thread_t * nil_find_thread (tstate_t state, void *p)

  Retrieves the highest priority thread in the specified state and associated to the specified object.
- static cnt_t nil_ready_all (void *p, cnt_t cnt, msg_t msg)

  Puts in ready state all thread matching the specified status and associated object.
- 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.
- void chSysInit (void)

  Initializes the kernel.
- void chSysHalt (const char *reason)

Halts the system.

- void chSysTimerHandlerI (void)

  Time management handler.
- void chSysUnconditionalLock (void)

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

  Unconditionally leaves the kernel lock state.
- 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.
- thread_t * chSchReadyI (thread_t *tp, msg_t msg)

  Makes the specified thread ready for execution.
- bool chSchIsPreemptionRequired (void)

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

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

  Reschedules if needed.
- msg_t chSchGoSleepTimeoutS (tstate_t newstate, sysinterval_t timeout)

  Puts the current thread to sleep into the specified state with timeout specification.
- 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 chThdResume (thread_reference_t *trp, msg_t msg)

  Wakes up a thread waiting on a thread reference object.
- void chThdSleep (sysinterval_t timeout)

  Suspends the invoking thread for the specified time.
- void chThdSleepUntil (systime_t abstime)

  Suspends the invoking thread until the system time arrives to the specified value.
- msg_t chThdEnqueueTimeoutS (threads_queue_t *tqp, sysinterval_t timeout)

  Enqueues the caller thread on a threads queue object.
- void chThdDoDequeueNextI (threads_queue_t *tqp, msg_t msg)

  Dequeues and wakes up one thread from the 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.
- 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 chSemReset (semaphore_t *sp, cnt_t n)

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

  Performs a reset operation on the semaphore.
- void chEvtSignal (thread_t *tp, eventmask_t mask)

  Adds a set of event flags directly to the specified `thread_t`.
- void chEvtSignalI (thread_t *tp, eventmask_t mask)

  Adds a set of event flags directly to the specified `thread_t`.
- eventmask_t chEvtWaitAnyTimeout (eventmask_t mask, sysinterval_t timeout)

  Waits for any of the specified events.

Variables

- nil_system_t nil

  System data structures.

6.3.2 Macro Definition Documentation

6.3.2.1 #define _CHIBIOS_NIL_

ChibiOS/NIL identification macro.

6.3.2.2 #define CH_KERNEL_STABLE 1

Stable release flag.

6.3.2.3 #define CH_KERNEL_VERSION "3.2.2"

Kernel version string.

6.3.2.4 #define CH_KERNEL_MAJOR 3

Kernel version major number.

6.3.2.5 #define CH_KERNEL_MINOR 2

Kernel version minor number.

6.3.2.6 #define CH_KERNEL_PATCH 2

Kernel version patch number.

6.3.2.7 #define FALSE 0

Generic 'false' preprocessor boolean constant.

Note

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

6.3.2.8 #define TRUE 1

Generic 'true' preprocessor boolean constant.

Note

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

6.3.2.9 #define MSG_OK (msg_t)0

OK wakeup message.

6.3.2.10 #define MSG_TIMEOUT (msg_t)-1

Wake-up caused by a timeout condition.

6.3.2.11 #define MSG_RESET (msg_t)-2

Wake-up caused by a reset condition.

6.3.2.12 #define TIME_IMMEDIATE ((sysinterval_t)-1)

Zero time specification for some functions with a timeout specification.

Note

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

6.3.2.13 #define TIME_INFINITE ((sysinterval_t)0)

Infinite time specification for all functions with a timeout specification.

6.3.2.14 #define TIME_MAX_INTERVAL ((sysinterval_t)-2)

Maximum interval constant usable as timeout.

6.3.2.15 #define TIME_MAX_SYSTIME ((systime_t)-1)

Maximum system of system time before it wraps.

6.3.2.16 #define NIL_STATE_READY (tstate_t)0

Thread ready or executing.

6.3.2.17 #define NIL_STATE_SLEEPING (tstate_t)1

Thread sleeping.

6.3.2.18 #define NIL_STATE_SUSP (tstate_t)2

Thread suspended.

6.3.2.19 #define NIL_STATE_WTQUEUE (tstate_t)3

On queue or semaph.

6.3.2.20 #define NIL_STATE_WTOREVT (tstate_t)4

Waiting for events.

6.3.2.21 #define ALL_EVENTS ((eventmask_t)-1)

All events allowed mask.

6.3.2.22 #define EVENT_MASK(*eid*) ((eventmask_t)(1 << (eid)))

Returns an event mask from an event identifier.

6.3.2.23 #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.

6.3.2.24 #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.

6.3.2.25 #define CH_CFG_FACTORY_OBJECTS_REGISTRY TRUE

Enables the registry of generic objects.

6.3.2.26 #define CH_CFG_FACTORY_GENERIC_BUFFERS TRUE

Enables factory for generic buffers.

6.3.2.27 #define CH_CFG_FACTORY_SEMAPHORES TRUE

Enables factory for semaphores.

6.3.2.28 #define CH_CFG_FACTORY_MAILBOXES TRUE

Enables factory for mailboxes.

6.3.2.29 #define CH_CFG_FACTORY_OBJ_FIFOS TRUE

Enables factory for objects FIFOs.

6.3.2.30 #define THD_IDLE_BASE (&__main_thread_stack_base__)

Boundaries of the idle thread boundaries, only required if stack checking is enabled.

6.3.2.31 #define __CH_STRINGIFY(*a*) #a

Utility to make the parameter a quoted string.

6.3.2.32 #define THD_TABLE_BEGIN const thread_config_t nil_thd_configs[CH_CFG_NUM_THREADS + 1] = {

Start of user threads table.

6.3.2.33 #define THD_TABLE_ENTRY(*wap, name, funcp, arg*)

Value:

```
{wap, ((stkalign_t *)(wap)) + (sizeof (wap) / sizeof(stkalign_t)),        \
   name, funcp, arg},
```

Entry of user threads table.

6.3.2.34 #define THD_TABLE_END

Value:

```
{THD_IDLE_BASE, THD_IDLE_END, "idle", NULL, NULL}                         \
};
```

End of user threads table.

6.3.2.35 #define MEM_ALIGN_MASK(*a*) ((size_t)(a) - 1U)

Alignment mask constant.

Parameters

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

6.3.2.36 #define MEM_ALIGN_PREV(*p, a*) ((size_t)(p) & ~MEM_ALIGN_MASK(a))

Aligns to the previous aligned memory address.

Parameters

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

6.3.2.37 #define MEM_ALIGN_NEXT(*p, a*)

Value:

```
MEM_ALIGN_PREV((size_t)(p) +                \
                                            MEM_ALIGN_MASK(a), (a))
```

Aligns to the new aligned memory address.

Parameters

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

6.3.2.38 #define MEM_IS_ALIGNED(*p*, *a*) (((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

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

Returns whatever a constant is a valid alignment.

Valid alignments are powers of two.

Parameters

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

6.3.2.40 #define THD_WORKING_AREA_SIZE(*n*)

Value:

```
MEM_ALIGN_NEXT(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.

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

Static working area allocation.

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

Parameters

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

Function Class:

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

6.3.2.42 #define THD_FUNCTION(*tname, arg*) PORT_THD_FUNCTION(tname, arg)

Thread declaration macro.

Note

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

6.3.2.43 #define CH_IRQ_IS_VALID_PRIORITY(*prio*) PORT_IRQ_IS_VALID_PRIORITY(prio)

Priority level validation macro.

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

Parameters

in	*prio*	the priority level

Returns

Priority range result.

Return values

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

6.3.2.44 #define CH_IRQ_IS_VALID_KERNEL_PRIORITY(*prio*) PORT_IRQ_IS_VALID_KERNEL_PRIORITY(prio)

Priority level validation macro.

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

Parameters

in	*prio*	the priority level

Returns

Priority range result.

Return values

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

6.3.2.45 #define CH_IRQ_PROLOGUE( )

Value:

```
PORT_IRQ_PROLOGUE();                                                    \
  _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.

6.3.2.46 #define CH_IRQ_EPILOGUE( )

Value:

```
_dbg_check_leave_isr();                                                 \
  PORT_IRQ_EPILOGUE()
```

IRQ handler exit code.

Note

Usually IRQ handlers function are also declared naked.

Function Class:

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

6.3.2.47 #define CH_IRQ_HANDLER(*id*) PORT_IRQ_HANDLER(id)

Standard normal IRQ handler declaration.

Note

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

Function Class:

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

6.3.2.48 #define CH_FAST_IRQ_HANDLER(*id*) PORT_FAST_IRQ_HANDLER(id)

Standard fast IRQ handler declaration.

Note

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

Function Class:

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

6.3.2.49 #define TIME_S2I(*secs*) ((sysinterval_t)((time_conv_t)(secs) * (time_conv_t)CH_CFG_ST_FREQUEN↩ CY))

Seconds to time interval.

Converts from seconds to system ticks number.

Note

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

Parameters

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.

6.3.2.50 #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.

6.3.2.51 #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.

6.3.2.52 #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.

6.3.2.53 #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.

6.3.2.54 #define TIME_I2US(*interval*)

Value:

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

Time interval to microseconds.

Converts from system ticks number to microseconds.

Note

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

Parameters

in	*interval*	interval in ticks

Returns

The number of microseconds.

Function Class:

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

6.3.2.55 #define _THREADS_QUEUE_DATA(*name*) {(cnt_t)0}

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

6.3.2.56 #define _THREADS_QUEUE_DECL(*name*) threads_queue_t name = _THREADS_QUEUE_DATA(name)

Static threads queue object initializer.

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

Parameters

in	*name*	the name of the threads queue variable

6.3.2.57 #define _SEMAPHORE_DATA(*name, n*) {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

6.3.2.58 #define SEMAPHORE_DECL(*name, n*) semaphore_t name = _SEMAPHORE_DATA(name, n)

Static semaphore initializer.

Statically initialized semaphores require no explicit initialization using `chSemInit()`.

Parameters

in	*name*	the name of the semaphore variable
in	*n*	the counter initial value, this value must be non-negative

6.3.2.59 #define chSysGetRealtimeCounterX( ) (rtcnt_t)port_rt_get_counter_value()

Returns the current value of the system real time counter.

Note

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

Returns

The value of the system realtime counter of type rtcnt_t.

Function Class:

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

6.3.2.60 #define chSysDisable( )

Value:

```
{                                                   \
  port_disable();                                                   \
  _dbg_check_disable();                                             \
}
```

Raises the system interrupt priority mask to the maximum level.

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

Note

Do not invoke this API from within a kernel lock.

Function Class:

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

6.3.2.61 #define chSysSuspend( )

Value:

```
{                                                   \
  port_suspend();                                                   \
  _dbg_check_suspend();                                             \
}
```

Raises the system interrupt priority mask to system level.

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

Note

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

Function Class:

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

6.3.2.62 #define chSysEnable( )

Value:

```
{                                                   \
  _dbg_check_enable();                                \
  port_enable();                                      \
}
```

Lowers the system interrupt priority mask to user level.

All the interrupt sources are enabled.

Note

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

Function Class:

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

6.3.2.63 #define chSysLock( )

Value:

```
{                                                   \
  port_lock();                                        \
  _dbg_check_lock();                                  \
}
```

Enters the kernel lock state.

Function Class:

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

6.3.2.64 #define chSysUnlock( )

Value:

```
{                                                   \
  _dbg_check_unlock();                                \
  port_unlock();                                      \
}
```

Leaves the kernel lock state.

Function Class:

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

6.3.2.65 #define chSysLockFromISR( )

Value:

```
{                                                   \
  port_lock_from_isr();                                                 \
  _dbg_check_lock_from_isr();                                           \
}
```

Enters the kernel lock state from within an interrupt handler.

Note

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

Function Class:

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

6.3.2.66 #define chSysUnlockFromISR( )

Value:

```
{                                                   \
  _dbg_check_unlock_from_isr();                                         \
  port_unlock_from_isr();                                               \
}
```

Leaves the kernel lock state from within an interrupt handler.

Note

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

Function Class:

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

6.3.2.67 #define chSchIsRescRequiredI( ) ((bool)(nil.current != nil.next))

Evaluates if a reschedule is required.

Return values

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

Function Class:

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

handlers.

6.3.2.68 #define chThdGetSelfX( ) nil.current

Returns a pointer to the current `thread_t`.

Function Class:

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

6.3.2.69 #define chThdSleepSeconds(*secs*) chThdSleep(TIME_S2I(secs))

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 clock.
The maximum specified value is implementation dependent.

Parameters

in	*secs*	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.

6.3.2.70 #define chThdSleepMilliseconds(*msecs*) chThdSleep(TIME_MS2I(msecs))

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 clock.
The maximum specified value is implementation dependent.

Parameters

in	*msecs*	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.

6.3.2.71 #define chThdSleepMicroseconds(*usecs*) chThdSleep(TIME_US2I(usecs))

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 clock.
The maximum specified value is implementation dependent.

Parameters

in	*usecs*	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.

6.3.2.72 #define chThdSleepS(*timeout*) (void) chSchGoSleepTimeoutS(NIL_STATE_SLEEPING, timeout)

Suspends the invoking thread for the specified time.

Parameters

in	*timeout*	the delay in system ticks

Function Class:

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

6.3.2.73 #define chThdSleepUntilS(*abstime*)

Value:

```
(void) chSchGoSleepTimeoutS(NIL_STATE_SLEEPING,
           \
                               chTimeDiffX(chVTGetSystemTimeX(), (abstime)))
```

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

Parameters

in	*abstime*	absolute system time

Function Class:

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

6.3.2.74 #define chThdQueueObjectInit(*tqp*) ((tqp)->cnt = (cnt_t)0)

Initializes a threads queue object.

Parameters

out	*tqp*	pointer to the threads queue object

Function Class:

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

6.3.2.75 #define chThdQueueIsEmptyI(*tqp*) ((bool)(tqp->cnt >= (cnt_t)0))

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.

6.3.2.76 #define chSemObjectInit(*sp, n*) ((sp)->cnt = (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:

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

6.3.2.77 #define chSemWait(*sp*) chSemWaitTimeout(sp, TIME_INFINITE)

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

CH_MSG_OK	if the thread has not stopped on the semaphore or the semaphore has been signaled.
CH_MSG_RST	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.

6.3.2.78 #define chSemWaitS(*sp*) chSemWaitTimeoutS(sp, TIME_INFINITE)

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

CH_MSG_OK	if the thread has not stopped on the semaphore or the semaphore has been signaled.
CH_MSG_RST	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.

6.3.2.79 #define chSemFastWaitI(*sp*) ((sp)->cnt--)

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.

6.3.2.80 #define chSemFastSignalI(*sp*) ((sp)->cnt++)

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.

6.3.2.81 #define chSemGetCounterI(*sp*) ((sp)->cnt)

Returns the semaphore counter current 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.

6.3.2.82 #define chVTGetSystemTimeX( ) (nil.systime)

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.

6.3.2.83 #define chVTTimeElapsedSinceX(*start*) chTimeDiffX((start), chVTGetSystemTimeX())

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.

6.3.2.84 #define chTimeAddX(*systime, interval*) ((systime_t)(systime) + (systime_t)(interval))

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.

6.3.2.85 #define chTimeDiffX(*start, end*) ((sysinterval_t)((systime_t)((systime_t)(end) - (systime_t)(start))))

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.

6.3.2.86 #define chTimeIsInRangeX(*time, start, end*)

Value:

```
((bool)((systime_t)((systime_t)(time) - (systime_t)(start)) <              \
          (systime_t)((systime_t)(end) - (systime_t)(start))))
```

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

Note

When start==end then the function returns always true because the whole time range is specified.

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	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.

6.3.2.87 #define chDbgCheck(*c*)

Value:

```
do {                                                                \
  /*lint -save -e506 -e774 [2.1, 14.3] Can be a constant by design.*/       \
  if (CH_DBG_ENABLE_CHECKS != FALSE) {                              \
    if (!(c)) {                                                     \
  /*lint -restore*/                                                 \
      chSysHalt(__func__);                                          \
    }                                                               \
  }                                                                 \
} while (false)
```

Function parameters check.

If the condition check fails then the kernel panics and halts.

Note

The condition is tested only if the CH_DBG_ENABLE_CHECKS switch is specified in chconf.h else the macro does nothing.

Parameters

in	*c*	the condition to be verified to be true

Function Class:

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

6.3.2.88 #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 (!(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.

6.3.3 Typedef Documentation

6.3.3.1 typedef uint32_t systime_t

Type of system time.

Note

It is selectable in configuration between 16 or 32 bits.

6.3.3.2 typedef uint32_t sysinterval_t

Type of time interval.

Note

It is selectable in configuration between 16 or 32 bits.

6.3.3.3 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.

6.3.3.4 typedef struct nil_thread thread_t

Type of a structure representing a thread.

Note

It is required as an early definition.

6.3.3.5 typedef struct nil_threads_queue threads_queue_t

Type of a queue of threads.

6.3.3.6 typedef threads_queue_t semaphore_t

Type of a structure representing a semaphore.

Note

Semaphores are implemented on thread queues, the object is the same, the behavior is slightly different.

6.3.3.7 typedef void(* tfunc_t) (void *p)

Thread function.

6.3.3.8 typedef struct nil_thread_cfg thread_config_t

Type of a structure representing a thread static configuration.

6.3.3.9 typedef thread_t* thread_reference_t

Type of a thread reference.

6.3.3.10 typedef struct nil_system nil_system_t

Type of a structure representing the system.

6.3.4 Function Documentation

6.3.4.1 static thread_t* nil_find_thread (tstate_t *state,* void * *p*) [static]

Retrieves the highest priority thread in the specified state and associated to the specified object.

Note

The search is unbounded, the thread is assumed to exist.

Parameters

in	*state*	thread state
in	*p*	object pointer

Returns

Pointer to the thread.

6.3.4.2 static cnt_t nil_ready_all (void * *p,* cnt_t *cnt,* msg_t *msg*) [static]

Puts in ready state all thread matching the specified status and associated object.

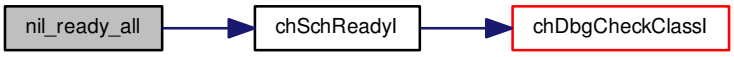
Parameters

in	*p*	object pointer
in	*cnt*	number of threads to be readied as a negative number, non negative numbers are ignored
in	*msg*	the wakeup message

Returns

Always zero.

Here is the call graph for this function:



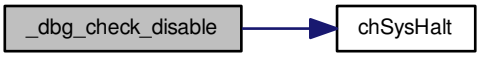
6.3.4.3 void _dbg_check_disable (void)

Guard code for `chSysDisable()`.

Function Class:

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

Here is the call graph for this function:



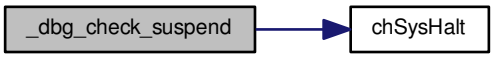
6.3.4.4 void _dbg_check_suspend (void)

Guard code for `chSysSuspend()`.

Function Class:

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

Here is the call graph for this function:



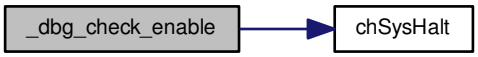
6.3.4.5 void _dbg_check_enable (void)

Guard code for `chSysEnable()`.

Function Class:

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

Here is the call graph for this function:



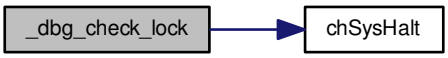
6.3.4.6 void _dbg_check_lock (void)

Guard code for `chSysLock()`.

Function Class:

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

Here is the call graph for this function:



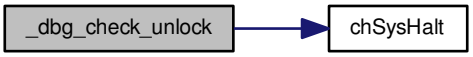
6.3.4.7 void _dbg_check_unlock (void)

Guard code for `chSysUnlock()`.

Function Class:

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

Here is the call graph for this function:



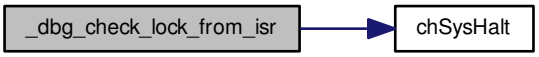
6.3.4.8 void _dbg_check_lock_from_isr (void)

Guard code for `chSysLockFromIsr()`.

Function Class:

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

Here is the call graph for this function:



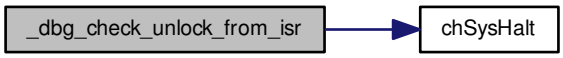
6.3.4.9 void _dbg_check_unlock_from_isr (void)

Guard code for `chSysUnlockFromIsr()`.

Function Class:

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

Here is the call graph for this function:



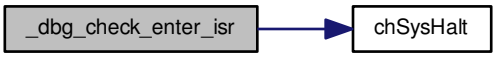
6.3.4.10 void _dbg_check_enter_isr (void)

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:



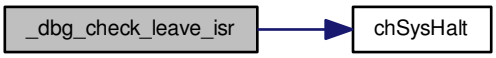
6.3.4.11 void _dbg_check_leave_isr (void)

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:



6.3.4.12 void chDbgCheckClassI (void)

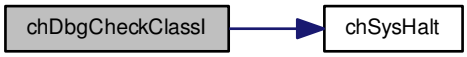
I-class functions context check.

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:



6.3.4.13 void chDbgCheckClassS (void)

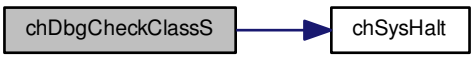
S-class functions context check.

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:



6.3.4.14 void chSysInit (void)

Initializes the kernel.

Initializes the kernel structures, the current instructions flow becomes the idle thread upon return. The idle thread must not invoke any kernel primitive able to change state to not runnable.

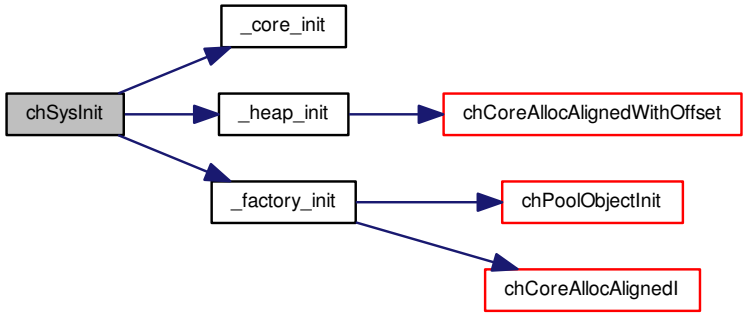
Note

This function assumes that the `nil` global variable has been zeroed by the runtime environment. If this is not the case then make sure to clear it before calling this function.

Function Class:

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

Here is the call graph for this function:



6.3.4.15 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.

6.3.4.16 void chSysTimerHandlerI (void)

Time management handler.

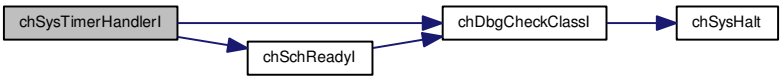
Note

This handler has to be invoked by a periodic ISR in order to reschedule the waiting threads.

Function Class:

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

Here is the call graph for this function:



6.3.4.17 void chSysUnconditionalLock (void)

Unconditionally enters the kernel lock state.

Note

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

Function Class:

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

6.3.4.18 void chSysUnconditionalUnlock (void)

Unconditionally leaves the kernel lock state.

Note

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

Function Class:

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

6.3.4.19 syssts_t chSysGetStatusAndLockX (void)

Returns the execution status and enters a critical zone.

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

Postcondition

The system is in a critical zone.

Returns

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

Function Class:

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

6.3.4.20 void chSysRestoreStatusX (syssts_t *sts*)

Restores the specified execution status and leaves a critical zone.

Note

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

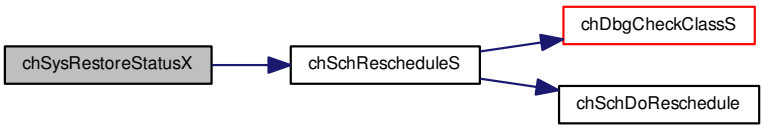
Parameters

in	*sts*	the system status to be restored.

Function Class:

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

Here is the call graph for this function:



6.3.4.21 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 true because the whole 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.

6.3.4.22 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`.

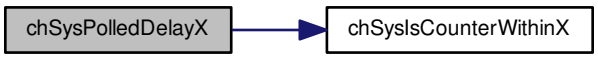
Parameters

`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:



6.3.4.23 thread_t * chSchReadyI (thread_t * *tp*, msg_t *msg*)

Makes the specified thread ready for execution.

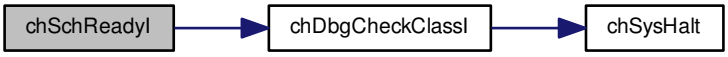
Parameters

`in`	*tp*	pointer to the `thread_t` object
`in`	*msg*	the wakeup message

Returns

The same reference passed as parameter.

Here is the call graph for this function:



6.3.4.24 bool chSchIsPreemptionRequired (void)

Evaluates if preemption is required.

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

Note

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

Return values

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.

6.3.4.25 void chSchDoReschedule (void)

Switches to the first thread on the runnable queue.

Note

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

Function Class:

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

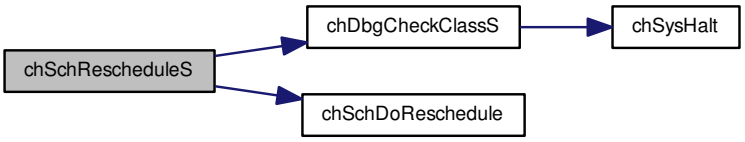
6.3.4.26 void chSchRescheduleS (void)

Reschedules if needed.

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:



6.3.4.27 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 system time then it is forcibly awakened with a `NIL_MSG_TMO` low level message.

Parameters

`in`	*newstate*	the new thread state or a semaphore pointer
`in`	*timeout*	the number of ticks before the operation timeouts. the following special values are allowed: • *TIME_INFINITE* no timeout.

Returns

The wakeup message.

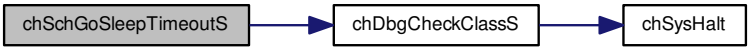
Return values

NIL_MSG_TMO	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:



6.3.4.28 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 number of ticks before the operation timeouts, the following special values are allowed: • *TIME_INFINITE* no timeout. • *TIME_IMMEDIATE* immediate timeout.

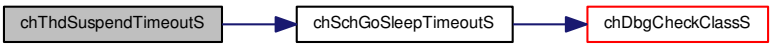
Returns

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:



6.3.4.29 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.

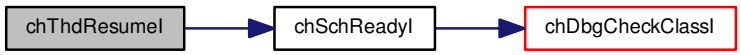
Parameters

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:



6.3.4.30 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.

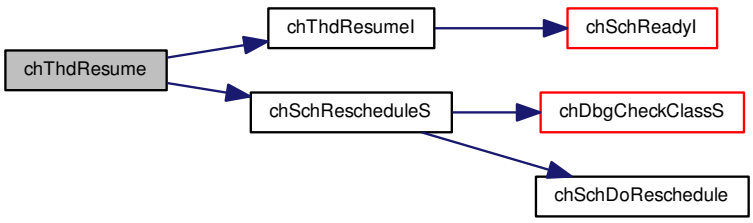
Parameters

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:



6.3.4.31 void chThdSleep (sysinterval_t *timeout*)

Suspends the invoking thread for the specified time.

Parameters

in	*timeout*	the delay in system ticks

Function Class:

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

6.3.4.32 void chThdSleepUntil (systime_t *abstime*)

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

Parameters

in	*abstime*	absolute system time

Function Class:

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

6.3.4.33 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 the threads queue object
in	*timeout*	the timeout in system ticks, the special values are handled as follow: • *TIME_INFINITE* no timeout. • *TIME_IMMEDIATE* immediate timeout.

Returns

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

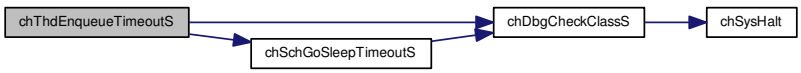
Return values

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:



6.3.4.34 void chThdDoDequeueNextI (threads_queue_t * *tqp*, msg_t *msg*)

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.

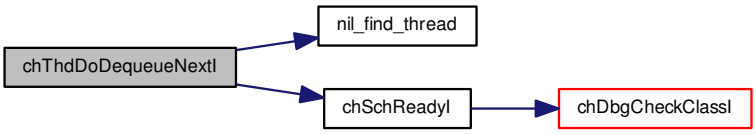
Parameters

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:



6.3.4.35 void chThdDequeueNextI (threads_queue_t * *tqp*, msg_t *msg*)

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

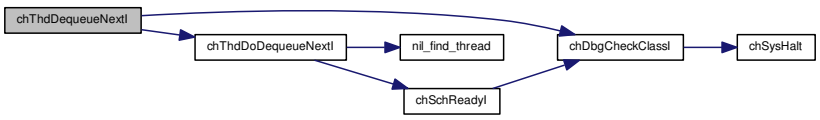
Parameters

in	*tqp*	pointer to the threads queue object
in	*msg*	the message code

Function Class:

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

Here is the call graph for this function:



6.3.4.36 void chThdDequeueAllI (threads_queue_t * *tqp*, msg_t *msg*)

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

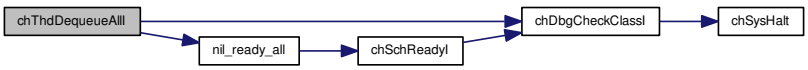
Parameters

in	*tqp*	pointer to 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:



6.3.4.37 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_INFINITE* no timeout. • *TIME_IMMEDIATE* immediate timeout.

Returns

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

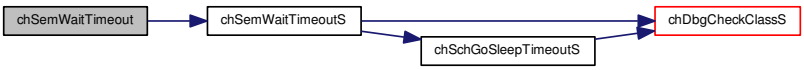
Return values

NIL_MSG_OK	if the thread has not stopped on the semaphore or the semaphore has been signaled.
NIL_MSG_RST	if the semaphore has been reset using `chSemReset()`.
NIL_MSG_TMO	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:



6.3.4.38 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_INFINITE* no timeout. • *TIME_IMMEDIATE* immediate timeout.

Returns

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

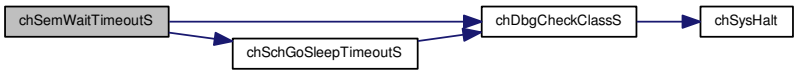
Return values

NIL_MSG_OK	if the thread has not stopped on the semaphore or the semaphore has been signaled.
NIL_MSG_RST	if the semaphore has been reset using `chSemReset()`.
NIL_MSG_TMO	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:



6.3.4.39 void chSemSignal (semaphore_t * *sp*)

Performs a signal operation on a semaphore.

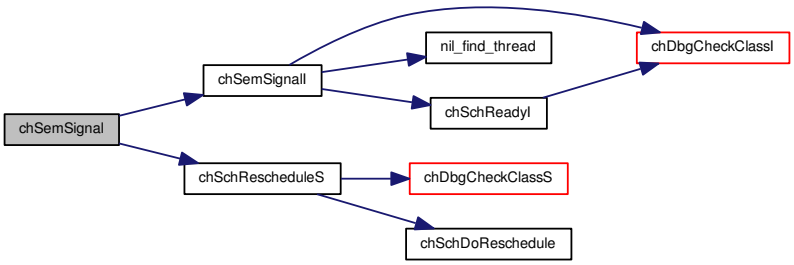
Parameters

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:



6.3.4.40 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.

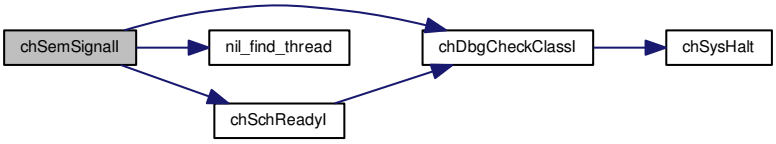
Parameters

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:



6.3.4.41 void chSemReset (semaphore_t * *sp*, cnt_t *n*)

Performs a reset operation on the semaphore.

Postcondition

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

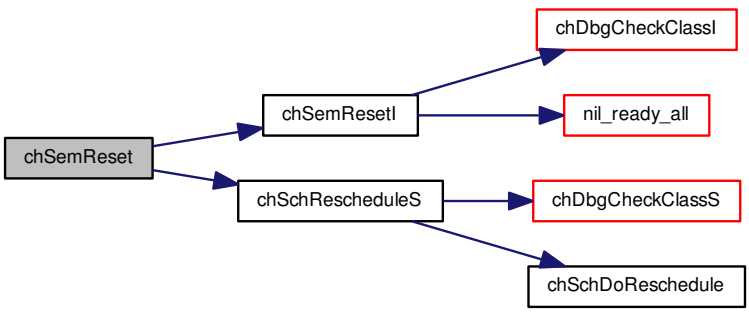
Parameters

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:



6.3.4.42 void chSemResetI (semaphore_t * *sp*, cnt_t *n*)

Performs a reset operation on the semaphore.

Postcondition

After invoking this function all the threads waiting on the semaphore, if any, are released and the semaphore counter is set to the specified, non negative, value.
This function does not reschedule so a call to a rescheduling function must be performed before unlocking the kernel. Note that interrupt handlers always reschedule on exit so an explicit reschedule must not be performed in ISRs.

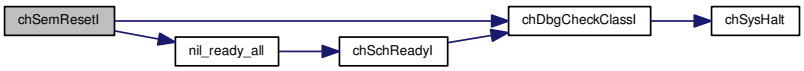
Parameters

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:



6.3.4.43 void chEvtSignal (thread_t * *tp*, eventmask_t *mask*)

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

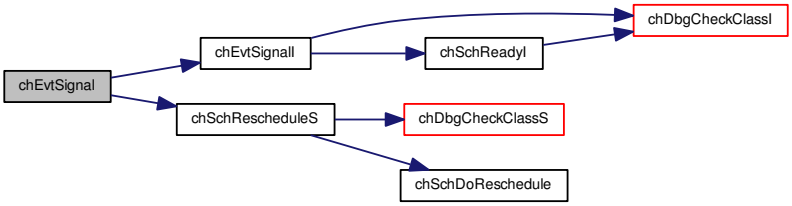
Parameters

in	*tp*	the thread to be signaled
in	*mask*	the event flags 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:



6.3.4.44 void chEvtSignalI (thread_t * *tp*, eventmask_t *mask*)

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.

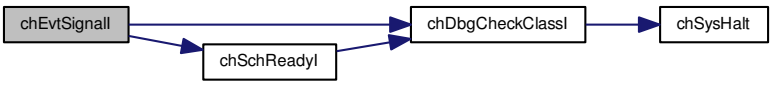
Parameters

`in`	*tp*	the thread to be signaled
`in`	*mask*	the event flags 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:



6.3.4.45 eventmask_t chEvtWaitAnyTimeout (eventmask_t *mask*, sysinterval_t *timeout*)

Waits for any of the specified events.

The function waits for any event among those specified in `mask` to become pending then the events are cleared and returned.

Parameters

in	*mask*	mask of the event flags 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_INFINITE* no timeout. • *TIME_IMMEDIATE* immediate timeout.

Returns

The mask of the served and cleared events.

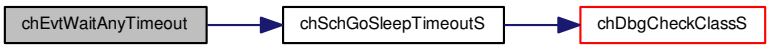
Return values

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:



6.3.5 Variable Documentation

6.3.5.1 nil_system_t nil

System data structures.

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.

Modules

- Version Numbers and Identification
- Synchronization
- Memory Management
- Complex Services

6.5 Version Numbers and Identification

6.5.1 Detailed Description

OS Library related info.

Macros

- #define _CHIBIOS_OSLIB_
  ChibiOS/LIB identification macro.
- #define CH_OSLIB_STABLE 1
  Stable release flag.

ChibiOS/LIB version identification

- #define CH_OSLIB_VERSION "1.1.2"
  OS Library version string.
- #define CH_OSLIB_MAJOR 1
  OS Library version major number.
- #define CH_OSLIB_MINOR 1
  OS Library version minor number.
- #define CH_OSLIB_PATCH 2
  OS Library version patch number.

6.5.2 Macro Definition Documentation

6.5.2.1 #define _CHIBIOS_OSLIB_

ChibiOS/LIB identification macro.

6.5.2.2 #define CH_OSLIB_STABLE 1

Stable release flag.

6.5.2.3 #define CH_OSLIB_VERSION "1.1.2"

OS Library version string.

6.5.2.4 #define CH_OSLIB_MAJOR 1

OS Library version major number.

6.5.2.5 #define CH_OSLIB_MINOR 1

OS Library version minor number.

6.5.2.6 #define CH_OSLIB_PATCH 2

OS Library version patch number.

6.6 Synchronization

6.6.1 Detailed Description

Synchronization services.

Modules

- Binary Semaphores
- Mailboxes
- Pipes

6.7 Binary Semaphores

6.7.1 Detailed Description

Macros

- #define _BSEMAPHORE_DATA(name, taken) {_SEMAPHORE_DATA(name.sem, ((taken) ? 0 : 1))}
  Data part of a static semaphore initializer.
- #define BSEMAPHORE_DECL(name, taken) binary_semaphore_t name = _BSEMAPHORE_DATA(name, taken)
  Static semaphore initializer.

Typedefs

- typedef struct ch_binary_semaphore binary_semaphore_t
  Binary semaphore type.

Data Structures

- struct ch_binary_semaphore
  Binary semaphore type.

Functions

- static void chBSemObjectInit (binary_semaphore_t *bsp, bool taken)
  Initializes a binary semaphore.
- static msg_t chBSemWait (binary_semaphore_t *bsp)
  Wait operation on the binary semaphore.
- static msg_t chBSemWaitS (binary_semaphore_t *bsp)
  Wait operation on the binary semaphore.
- static msg_t chBSemWaitTimeoutS (binary_semaphore_t *bsp, sysinterval_t timeout)
  Wait operation on the binary semaphore.
- static msg_t chBSemWaitTimeout (binary_semaphore_t *bsp, sysinterval_t timeout)
  Wait operation on the binary semaphore.
- static void chBSemResetI (binary_semaphore_t *bsp, bool taken)
  Reset operation on the binary semaphore.
- static void chBSemReset (binary_semaphore_t *bsp, bool taken)
  Reset operation on the binary semaphore.
- static void chBSemSignalI (binary_semaphore_t *bsp)
  Performs a signal operation on a binary semaphore.
- static void chBSemSignal (binary_semaphore_t *bsp)
  Performs a signal operation on a binary semaphore.
- static bool chBSemGetStateI (const binary_semaphore_t *bsp)
  Returns the binary semaphore current state.

6.7.2 Macro Definition Documentation

6.7.2.1 #define _BSEMAPHORE_DATA(*name, taken*) {_SEMAPHORE_DATA(name.sem, ((taken) ? 0 : 1))}

Data part of a static semaphore initializer.

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

Parameters

in	*name*	the name of the semaphore variable
in	*taken*	the semaphore initial state

6.7.2.2 #define BSEMAPHORE_DECL(*name, taken*) binary_semaphore_t name = _BSEMAPHORE_DATA(name, taken)

Static semaphore initializer.

Statically initialized semaphores require no explicit initialization using `chBSemInit()`.

Parameters

in	*name*	the name of the semaphore variable
in	*taken*	the semaphore initial state

6.7.3 Typedef Documentation

6.7.3.1 typedef struct ch_binary_semaphore binary_semaphore_t

Binary semaphore type.

6.7.4 Function Documentation

6.7.4.1 static 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:

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

6.7.4.2 static msg_t chBSemWait (binary_semaphore_t * *bsp*) [inline], [static]

Wait operation on the binary semaphore.

Parameters

in	*bsp*	pointer to a `binary_semaphore_t` structure

Returns

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

Return values

MSG_OK	if the binary semaphore has been successfully taken.
MSG_RESET	if the binary semaphore has been reset using `bsemReset()`.

Function Class:

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

6.7.4.3 static msg_t chBSemWaitS (binary_semaphore_t * *bsp*) [inline], [static]

Wait operation on the binary semaphore.

Parameters

`in`	*bsp*	pointer to a `binary_semaphore_t` structure

Returns

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

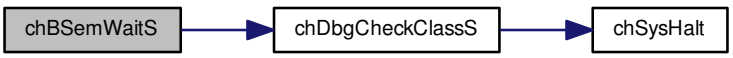
Return values

MSG_OK	if the binary semaphore has been successfully taken.
MSG_RESET	if the binary semaphore has been reset using `bsemReset()`.

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:



6.7.4.4 static 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.

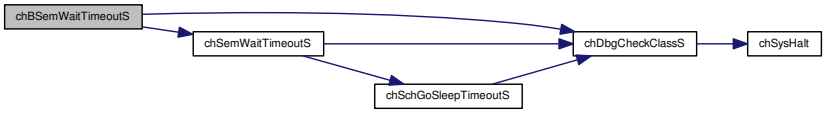
Return values

MSG_OK	if the binary semaphore has been successfully taken.
MSG_RESET	if the binary semaphore has been reset using `bsemReset()`.
MSG_TIMEOUT	if the binary semaphore has not been signaled or reset within the specified timeout.

Function Class:

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

Here is the call graph for this function:



6.7.4.5 static 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.

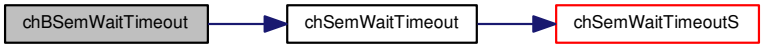
Return values

MSG_OK	if the binary semaphore has been successfully taken.
MSG_RESET	if the binary semaphore has been reset using `bsemReset()`.
MSG_TIMEOUT	if the binary semaphore has not been signaled or reset within the specified timeout.

Function Class:

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

Here is the call graph for this function:



6.7.4.6 static void chBSemResetI (binary_semaphore_t * *bsp*, bool *taken*) [inline], [static]

Reset operation on the binary semaphore.

Note

The released threads can recognize they were waked up by a reset rather than a signal because the `bsem↩Wait()` will return `MSG_RESET` instead of `MSG_OK`.
This function does not reschedule.

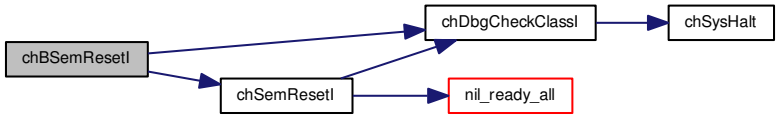
Parameters

`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:



6.7.4.7 static void chBSemReset (binary_semaphore_t * *bsp*, bool *taken*) [inline], [static]

Reset operation on the binary semaphore.

Note

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

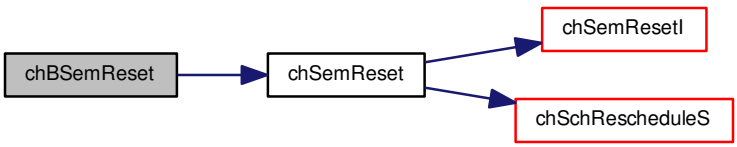
Parameters

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:



6.7.4.8 static void chBSemSignalI (binary_semaphore_t * *bsp*) [inline], [static]

Performs a signal operation on a binary semaphore.

Note

This function does not reschedule.

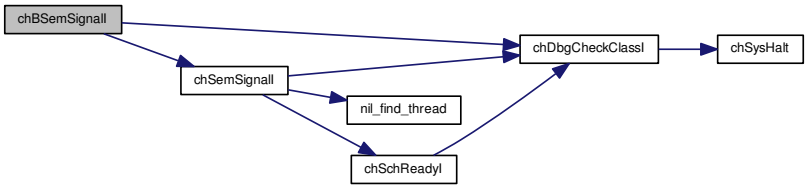
Parameters

in	*bsp*	pointer to a `binary_semaphore_t` structure

Function Class:

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

Here is the call graph for this function:



6.7.4.9 static void chBSemSignal (binary_semaphore_t * *bsp*) [inline], [static]

Performs a signal operation on a binary semaphore.

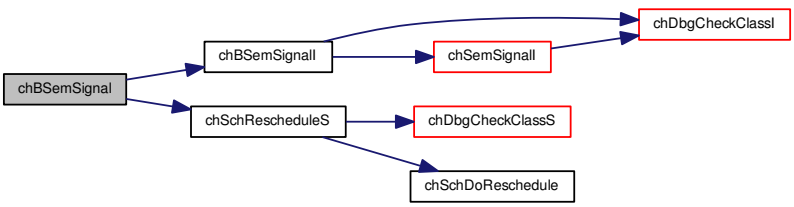
Parameters

in	*bsp*	pointer to a `binary_semaphore_t` structure

Function Class:

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

Here is the call graph for this function:



6.7.4.10 static bool chBSemGetStateI (const binary_semaphore_t * *bsp*) [inline], [static]

Returns the binary semaphore current state.

Parameters

in	*bsp*	pointer to a `binary_semaphore_t` structure

Returns

The binary semaphore current state.

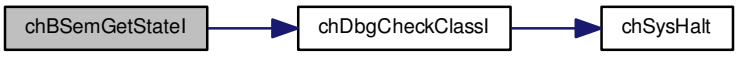
Return values

false	if the binary semaphore is not taken.
true	if the binary semaphore is taken.

Function Class:

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

Here is the call graph for this function:



6.8 Mailboxes

6.8.1 Detailed Description

Asynchronous messages.

Operation mode

A mailbox is an asynchronous communication mechanism.
Operations defined for mailboxes:

- **Post**: Posts a message on the mailbox in FIFO order.
- **Post Ahead**: Posts a message on the mailbox with urgent priority.
- **Fetch**: A message is fetched from the mailbox and removed from the queue.
- **Reset**: The mailbox is emptied and all the stored messages are lost.

A message is a variable of type msg_t that is guaranteed to have the same size of and be compatible with (data) pointers (anyway an explicit cast is needed). If larger messages need to be exchanged then a pointer to a structure can be posted in the mailbox but the posting side has no predefined way to know when the message has been processed. A possible approach is to allocate memory (from a memory pool for example) from the posting side and free it on the fetching side. Another approach is to set a "done" flag into the structure pointed by the message.

Precondition

In order to use the mailboxes APIs the `CH_CFG_USE_MAILBOXES` option must be enabled in `chconf.h`.

Note

Compatible with RT and NIL.

Macros

- #define _MAILBOX_DATA(name, buffer, size)

  Data part of a static mailbox initializer.
- #define MAILBOX_DECL(name, buffer, size) mailbox_t name = _MAILBOX_DATA(name, buffer, size)

  Static mailbox initializer.

Data Structures

- struct mailbox_t

  Structure representing a mailbox object.

Functions

- void chMBObjectInit (mailbox_t *mbp, msg_t *buf, size_t n)

  Initializes a `mailbox_t` object.
- void chMBReset (mailbox_t *mbp)

  Resets a `mailbox_t` object.
- void chMBResetI (mailbox_t *mbp)

  Resets a `mailbox_t` object.
- msg_t chMBPostTimeout (mailbox_t *mbp, msg_t msg, sysinterval_t timeout)

Posts a message into a mailbox.

- msg_t chMBPostTimeoutS (mailbox_t *mbp, msg_t msg, sysinterval_t timeout)

  Posts a message into a mailbox.
- msg_t chMBPostI (mailbox_t *mbp, msg_t msg)

  Posts a message into a mailbox.
- msg_t chMBPostAheadTimeout (mailbox_t *mbp, msg_t msg, sysinterval_t timeout)

  Posts an high priority message into a mailbox.
- msg_t chMBPostAheadTimeoutS (mailbox_t *mbp, msg_t msg, sysinterval_t timeout)

  Posts an high priority message into a mailbox.
- msg_t chMBPostAheadI (mailbox_t *mbp, msg_t msg)

  Posts an high priority message into a mailbox.
- msg_t chMBFetchTimeout (mailbox_t *mbp, msg_t *msgp, sysinterval_t timeout)

  Retrieves a message from a mailbox.
- msg_t chMBFetchTimeoutS (mailbox_t *mbp, msg_t *msgp, sysinterval_t timeout)

  Retrieves a message from a mailbox.
- msg_t chMBFetchI (mailbox_t *mbp, msg_t *msgp)

  Retrieves a message from a mailbox.
- static size_t chMBGetSizeI (const mailbox_t *mbp)

  Returns the mailbox buffer size as number of messages.
- static size_t chMBGetUsedCountI (const mailbox_t *mbp)

  Returns the number of used message slots into a mailbox.
- static size_t chMBGetFreeCountI (const mailbox_t *mbp)

  Returns the number of free message slots into a mailbox.
- static msg_t chMBPeekI (const mailbox_t *mbp)

  Returns the next message in the queue without removing it.
- static void chMBResumeX (mailbox_t *mbp)

  Terminates the reset state.

6.8.2 Macro Definition Documentation

6.8.2.1 #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

6.8.2.2 #define MAILBOX_DECL(*name, buffer, size*) mailbox_t name = _MAILBOX_DATA(name, buffer, size)

Static mailbox initializer.

Statically initialized mailboxes require no explicit initialization using `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

6.8.3 Function Documentation

6.8.3.1 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:

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

6.8.3.2 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()`.

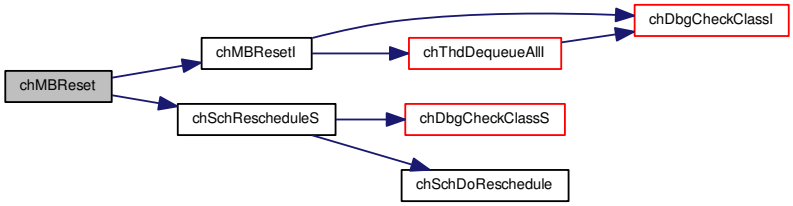
Parameters

`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:



6.8.3.3 void chMBResetI (mailbox_t * *mbp*)

Resets a `mailbox_t` object.

All the waiting threads are resumed with status `MSG_RESET` and the queued messages are lost.

Postcondition

The mailbox is in reset state, all operations will fail and return `MSG_RESET` until the mailbox is enabled again using `chMBResumeX()`.

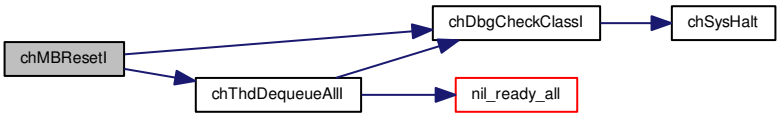
Parameters

`in`	*mbp*	the pointer to an initialized `mailbox_t` object

Function Class:

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

Here is the call graph for this function:



6.8.3.4 msg_t chMBPostTimeout (mailbox_t * *mbp*, msg_t *msg*, sysinterval_t *timeout*)

Posts a message into a mailbox.

The invoking thread waits until 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

Parameters

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.

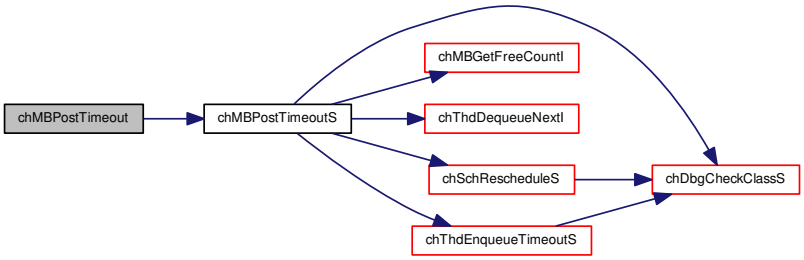
Return values

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:



6.8.3.5 msg_t chMBPostTimeoutS (mailbox_t * *mbp*, msg_t *msg*, sysinterval_t *timeout*)

Posts a message into a mailbox.

The invoking thread waits until a empty slot in the mailbox becomes available or the specified time runs out.

Parameters

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.

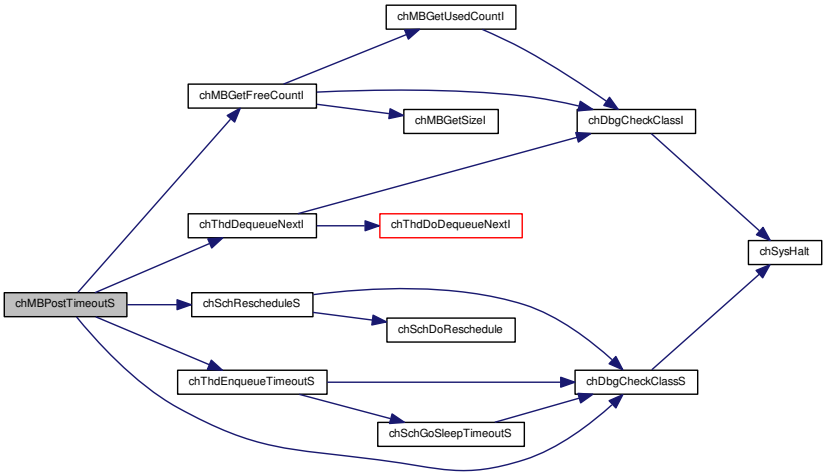
Return values

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:



6.8.3.6 msg_t chMBPostI (mailbox_t * *mbp*, msg_t *msg*)

Posts a message into a mailbox.

This variant is non-blocking, the function returns a timeout condition if the queue is full.

Parameters

in	*mbp*	the pointer to an initialized `mailbox_t` object
in	*msg*	the message to be posted on the mailbox

Returns

The operation status.

Return values

MSG_OK	if a message has been correctly posted.

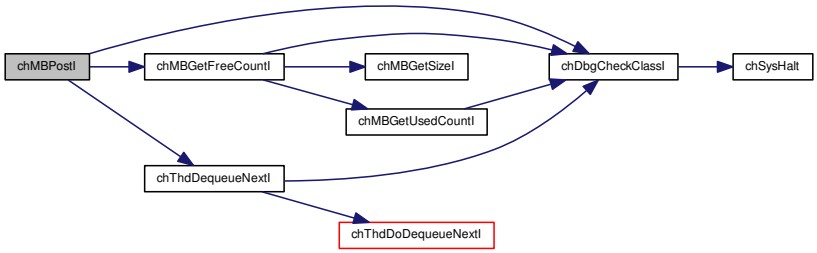
Return values

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:



6.8.3.7 msg_t chMBPostAheadTimeout (mailbox_t * *mbp*, msg_t *msg*, sysinterval_t *timeout*)

Posts an high priority message into a mailbox.

The invoking thread waits until a empty slot in the mailbox becomes available or the specified time runs out.

Parameters

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.

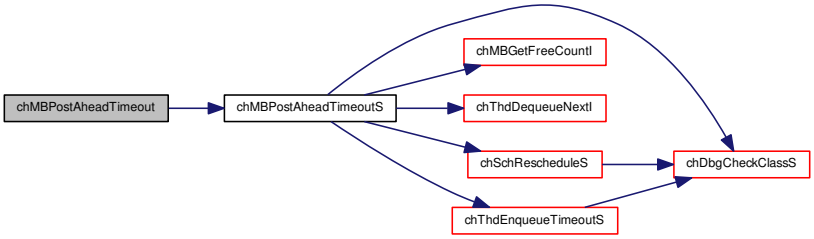
Return values

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:



6.8.3.8 msg_t chMBPostAheadTimeoutS (mailbox_t * *mbp*, msg_t *msg*, sysinterval_t *timeout*)

Posts an high priority message into a mailbox.

The invoking thread waits until 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.

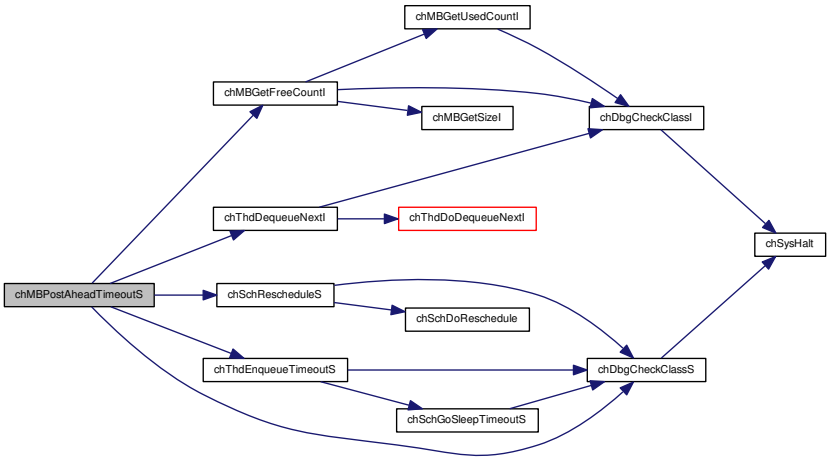
Return values

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:



6.8.3.9 msg_t chMBPostAheadI (mailbox_t * *mbp*, msg_t *msg*)

Posts an high priority message into a mailbox.

This variant is non-blocking, the function returns a timeout condition if the queue is full.

Parameters

in	*mbp*	the pointer to an initialized mailbox_t object
in	*msg*	the message to be posted on the mailbox

Returns

The operation status.

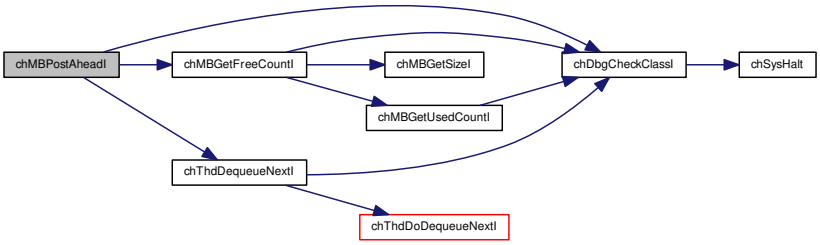
Return values

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:



6.8.3.10 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.

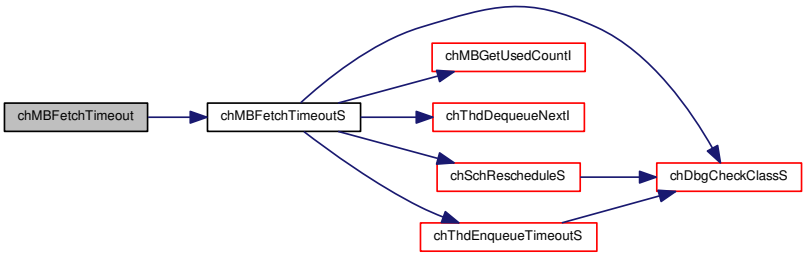
Return values

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:



6.8.3.11 msg_t chMBFetchTimeoutS (mailbox_t * *mbp*, msg_t * *msgp*, sysinterval_t *timeout*)

Retrieves a message from a mailbox.

The invoking thread waits until a message is posted in the mailbox or the specified time runs out.

Parameters

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.

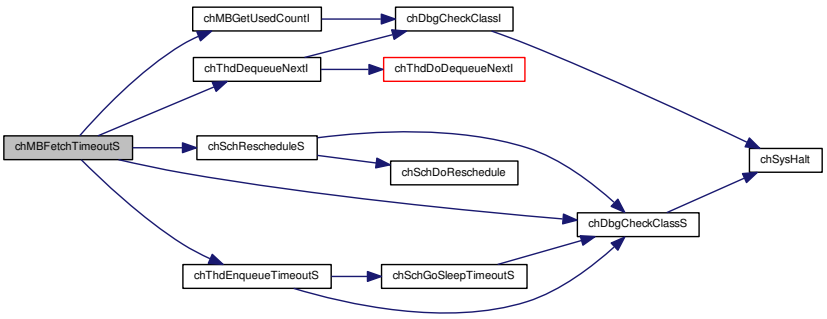
Return values

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:



6.8.3.12 msg_t chMBFetchI (mailbox_t * *mbp*, msg_t * *msgp*)

Retrieves a message from a mailbox.

This variant is non-blocking, the function returns a timeout condition if the queue is empty.

Parameters

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.

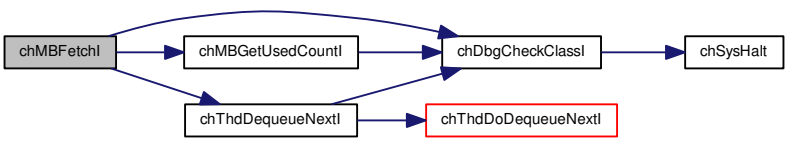
Return values

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:



6.8.3.13 static size_t chMBGetSizeI (const mailbox_t * *mbp*) [inline], [static]

Returns the mailbox buffer size as number of messages.

Parameters

in	*mbp*	the pointer to an initialized mailbox_t object

Returns

The size of the mailbox.

Function Class:

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

6.8.3.14 static size_t chMBGetUsedCountI (const mailbox_t * *mbp*) [inline], [static]

Returns the number of used message slots into a mailbox.

Parameters

in	*mbp*	the pointer to an initialized mailbox_t object

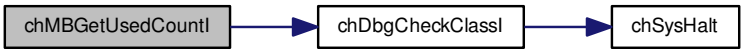
Returns

The number of queued messages.

Function Class:

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

Here is the call graph for this function:



6.8.3.15 static size_t chMBGetFreeCountI (const mailbox_t * *mbp*) [inline], [static]

Returns the number of free message slots into a mailbox.

Parameters

in	*mbp*	the pointer to an initialized mailbox_t object

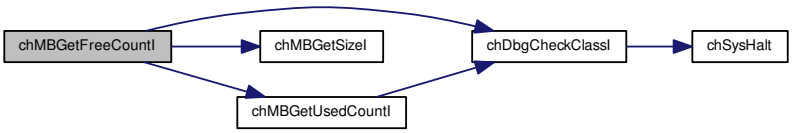
Returns

The number of empty message slots.

Function Class:

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

Here is the call graph for this function:



6.8.3.16 static msg_t chMBPeekI (const mailbox_t * *mbp*) [inline], [static]

Returns the next message in the queue without removing it.

Precondition

A message must be waiting in the queue for this function to work or it would return garbage. The correct way to use this macro is to use `chMBGetUsedCountI()` and then use this macro, all within a lock state.

Parameters

in	*mbp*	the pointer to an initialized `mailbox_t` object

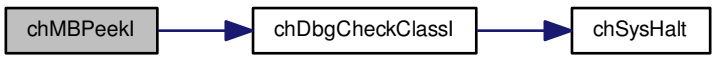
Returns

The next message in queue.

Function Class:

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

Here is the call graph for this function:



6.8.3.17 static void chMBResumeX (mailbox_t * *mbp*) [inline], [static]

Terminates the reset state.

Parameters

in	*mbp*	the pointer to an initialized mailbox_t object

Function Class:

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

6.9 Pipes

6.9.1 Detailed Description

Macros

- #define _PIPE_DATA(name, buffer, size)

  Data part of a static pipe initializer.
- #define PIPE_DECL(name, buffer, size) pipe_t name = _PIPE_DATA(name, buffer, size)

  Static pipe initializer.

Data Structures

- struct pipe_t

  Structure representing a pipe object.

Functions

- static size_t pipe_write (pipe_t *pp, const uint8_t *bp, size_t n)

  Non-blocking pipe write.
- static size_t pipe_read (pipe_t *pp, uint8_t *bp, size_t n)

  Non-blocking pipe read.
- void chPipeObjectInit (pipe_t *pp, uint8_t *buf, size_t n)

  Initializes a `mailbox_t` object.
- void chPipeReset (pipe_t *pp)

  Resets a `pipe_t` object.
- size_t chPipeWriteTimeout (pipe_t *pp, const uint8_t *bp, size_t n, sysinterval_t timeout)

  Pipe write with timeout.
- size_t chPipeReadTimeout (pipe_t *pp, uint8_t *bp, size_t n, sysinterval_t timeout)

  Pipe read with timeout.
- static size_t chPipeGetSize (const pipe_t *pp)

  Returns the pipe buffer size as number of bytes.
- static size_t chPipeGetUsedCount (const pipe_t *pp)

  Returns the number of used byte slots into a pipe.
- static size_t chPipeGetFreeCount (const pipe_t *pp)

  Returns the number of free byte slots into a pipe.
- static void chPipeResume (pipe_t *pp)

  Terminates the reset state.

6.9.2 Macro Definition Documentation

6.9.2.1 #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

6.9.2.2 #define PIPE_DECL(*name, buffer, size*) pipe_t name = _PIPE_DATA(name, buffer, size)

Static pipe initializer.

Statically initialized pipes require no explicit initialization using `chPipeObjectInit()`.

Parameters

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

6.9.3 Function Documentation

6.9.3.1 static size_t pipe_write (pipe_t * *pp*, const uint8_t * *bp*, size_t *n*) [static]

Non-blocking pipe write.

The function writes data from a buffer to a pipe. The operation completes when the specified amount of data has been transferred or when the pipe buffer has been filled.

Parameters

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

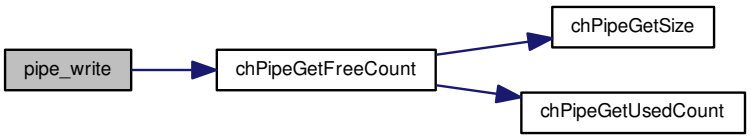
Returns

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:



6.9.3.2 static size_t pipe_read (pipe_t * *pp*, uint8_t * *bp*, size_t *n*) [static]

Non-blocking pipe read.

The function reads data from a pipe into a buffer. The operation completes when the specified amount of data has been transferred or when the pipe buffer has been emptied.

Parameters

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

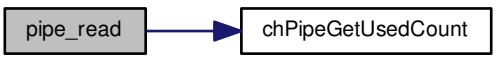
Returns

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:



6.9.3.3 void chPipeObjectInit (pipe_t * *pp*, uint8_t * *buf*, size_t *n*)

Initializes a `mailbox_t` object.

Parameters

`out`	*pp*	the pointer to the `pipe_t` structure to be initialized
`in`	*buf*	pointer to the pipe buffer as an array of `uint8_t`
`in`	*n*	number of elements in the buffer array

Function Class:

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

6.9.3.4 void chPipeReset (pipe_t * *pp*)

Resets a `pipe_t` object.

All the waiting threads are resumed with status `MSG_RESET` and the queued data is lost.

Postcondition

The pipe is in reset state, all operations will fail and return `MSG_RESET` until the mailbox is enabled again using `chPipeResumeX()`.

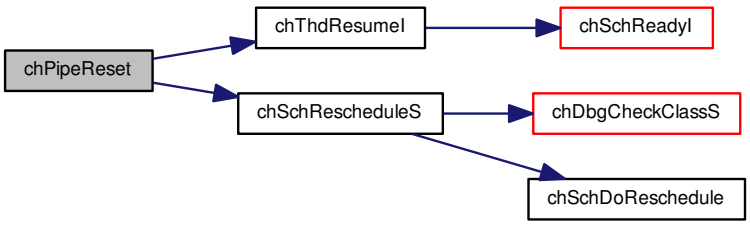
Parameters

`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:



6.9.3.5 size_t chPipeWriteTimeout (pipe_t * *pp*, const uint8_t * *bp*, size_t *n*, sysinterval_t *timeout*)

Pipe write with timeout.

The function writes data from a buffer to a pipe. The operation completes when the specified amount of data has been transferred or after the specified timeout or if the pipe has been reset.

Parameters

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.

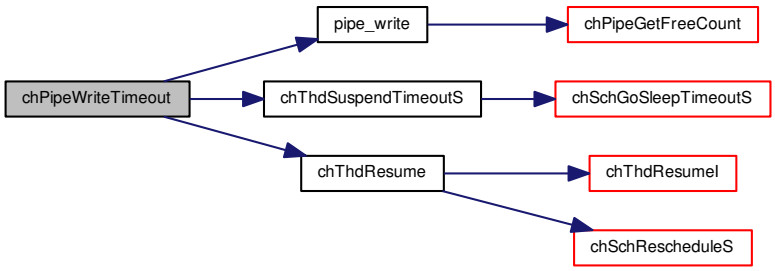
Returns

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:



6.9.3.6 size_t chPipeReadTimeout (pipe_t * *pp*, uint8_t * *bp*, size_t *n*, sysinterval_t *timeout*)

Pipe read with timeout.

The function reads data from a pipe into a buffer. The operation completes when the specified amount of data has been transferred or after the specified timeout or if the pipe has been reset.

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.

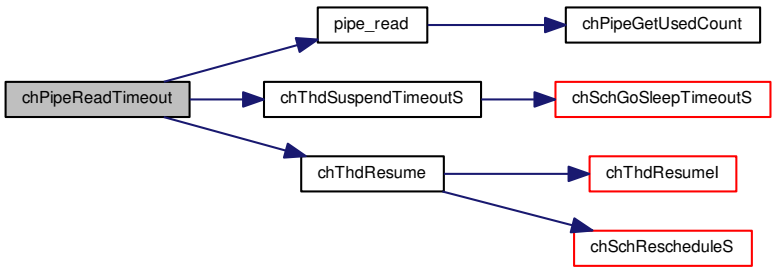
Returns

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:



6.9.3.7 static size_t chPipeGetSize (const pipe_t * *pp*) [inline], [static]

Returns the pipe buffer size as number of bytes.

Parameters

in	*pp*	the pointer to an initialized `pipe_t` object

Returns

The size of the pipe.

Function Class:

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

6.9.3.8 static size_t chPipeGetUsedCount (const pipe_t * *pp*) [inline], [static]

Returns the number of used byte slots into a pipe.

Parameters

in	*pp*	the pointer to an initialized `pipe_t` object

Returns

The number of queued bytes.

Function Class:

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

6.9.3.9 static size_t chPipeGetFreeCount (const pipe_t * *pp*) [inline], [static]

Returns the number of free byte slots into a pipe.

Parameters

in	*pp*	the pointer to an initialized pipe_t object

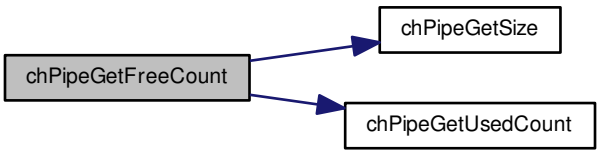
Returns

The number of empty byte slots.

Function Class:

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

Here is the call graph for this function:



6.9.3.10 static void chPipeResume (pipe_t * *pp*) [inline], [static]

Terminates the reset state.

Parameters

in	*pp*	the pointer to an initialized pipe_t object

Function Class:

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

6.10 Memory Management

6.10.1 Detailed Description

Memory Management services.

Modules

- Core Memory Manager
- Memory Heaps
- Memory Pools

6.11 Core Memory Manager

6.11.1 Detailed Description

Core Memory Manager related APIs and services.

Operation mode

The core memory manager is a simplified allocator that only allows to allocate memory blocks without the possibility to free them.
This allocator is meant as a memory blocks provider for the other allocators such as:

- C-Runtime allocator (through a compiler specific adapter module).
- Heap allocator (see Memory Heaps).
- Memory pools allocator (see Memory Pools).

By having a centralized memory provider the various allocators can coexist and share the main memory.
This allocator, alone, is also useful for very simple applications that just require a simple way to get memory blocks.

Precondition

In order to use the core memory manager APIs the `CH_CFG_USE_MEMCORE` option must be enabled in `chconf.h`.

Note

Compatible with RT and NIL.

Macros

- #define CH_CFG_MEMCORE_SIZE 0

  Managed RAM size.

Typedefs

- typedef void *(* memgetfunc_t) (size_t size, unsigned align)

  Memory get function.
- typedef void *(* memgetfunc2_t) (size_t size, unsigned align, size_t offset)

  Enhanced memory get function.

Data Structures

- struct memcore_t

  Type of memory core object.

Functions

- void _core_init (void)

  Low level memory manager initialization.
- void * chCoreAllocAlignedWithOffsetI (size_t size, unsigned align, size_t offset)

  Allocates a memory block.

- void * chCoreAllocAlignedWithOffset (size_t size, unsigned align, size_t offset)

  Allocates a memory block.
- size_t chCoreGetStatusX (void)

  Core memory status.
- static void * chCoreAllocAlignedI (size_t size, unsigned align)

  Allocates a memory block.
- static void * chCoreAllocAligned (size_t size, unsigned align)

  Allocates a memory block.
- static void * chCoreAllocI (size_t size)

  Allocates a memory block.
- static void * chCoreAlloc (size_t size)

  Allocates a memory block.

Variables

- memcore_t ch_memcore

  Memory core descriptor.

6.11.2 Macro Definition Documentation

6.11.2.1 #define CH_CFG_MEMCORE_SIZE 0

Managed RAM size.

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

Note

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

6.11.3 Typedef Documentation

6.11.3.1 typedef void*(* memgetfunc_t) (size_t size, unsigned align)

Memory get function.

6.11.3.2 typedef void*(* memgetfunc2_t) (size_t size, unsigned align, size_t offset)

Enhanced memory get function.

6.11.4 Function Documentation

6.11.4.1 void _core_init (void)

Low level memory manager initialization.

Function Class:

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

6.11.4.2 void * chCoreAllocAlignedWithOffsetI (size_t *size*, unsigned *align*, size_t *offset*)

Allocates a memory block.

This function allocates a block of `offset + size` bytes. The returned pointer has `offset` bytes before its address and `size` bytes after.

Parameters

`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.

Here is the call graph for this function:

6.11.4.3 void * chCoreAllocAlignedWithOffset (size_t *size*, unsigned *align*, size_t *offset*)

Allocates a memory block.

This function allocates a block of `offset + size` bytes. The returned pointer has `offset` bytes before its address and `size` bytes after.

Parameters

`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.

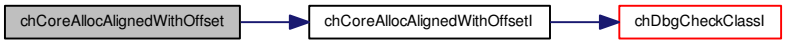
Return values

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:



6.11.4.4 size_t chCoreGetStatusX (void)

Core memory status.

Returns

The size, in bytes, of the free core memory.

Function Class:

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

6.11.4.5 static void* chCoreAllocAlignedI (size_t *size*, unsigned *align*) [inline], [static]

Allocates a memory block.

The allocated block is guaranteed to be properly aligned to the specified alignment.

Parameters

in	*size*	the size of the block to be allocated.
in	*align*	desired memory alignment

Returns

A pointer to the allocated memory block.

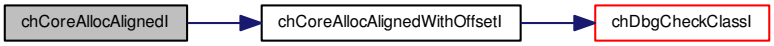
Return values

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.

Here is the call graph for this function:



6.11.4.6 static void* chCoreAllocAligned (size_t *size*, unsigned *align*) [inline], [static]

Allocates a memory block.

The allocated block is guaranteed to be properly aligned to the specified alignment.

Parameters

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.

Here is the call graph for this function:

6.11.4.7 static void* chCoreAllocI (size_t *size*) [inline], [static]

Allocates a memory block.

The allocated block is guaranteed to be properly aligned for a pointer data type.

Parameters

in	*size*	the size of the block to be allocated.

Returns

A pointer to the allocated memory block.

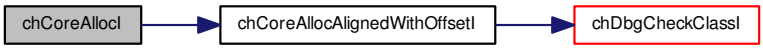
Return values

NULL	allocation failed, core memory exhausted.

Function Class:

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

Here is the call graph for this function:



6.11.4.8 static void* chCoreAlloc (size_t *size*) [inline], [static]

Allocates a memory block.

The allocated block is guaranteed to be properly aligned for a pointer data type.

Parameters

in	*size*	the size of the block to be allocated.

Returns

A pointer to the allocated memory block.

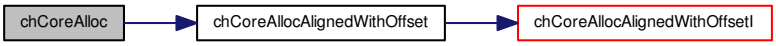
Return values

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.

Here is the call graph for this function:



6.11.5 Variable Documentation

6.11.5.1 memcore_t ch_memcore

Memory core descriptor.

6.12 Memory Heaps

6.12.1 Detailed Description

Heap Allocator related APIs.

Operation mode

The heap allocator implements a first-fit strategy and its APIs are functionally equivalent to the usual `malloc()` and `free()` library functions. The main difference is that the OS heap APIs are guaranteed to be thread safe and there is the ability to return memory blocks aligned to arbitrary powers of two.

Precondition

In order to use the heap APIs the `CH_CFG_USE_HEAP` option must be enabled in `chconf.h`.

Note

Compatible with RT and NIL.

Macros

- #define CH_HEAP_ALIGNMENT 8U

  Minimum alignment used for heap.
- #define CH_HEAP_AREA(name, size)

  Allocation of an aligned static heap buffer.

Typedefs

- typedef struct memory_heap memory_heap_t

  Type of a memory heap.
- typedef union heap_header heap_header_t

  Type of a memory heap header.

Data Structures

- union heap_header

  Memory heap block header.
- struct memory_heap

  Structure describing a memory heap.

Functions

- void _heap_init (void)

  Initializes the default heap.
- void chHeapObjectInit (memory_heap_t *heapp, void *buf, size_t size)

  Initializes a memory heap from a static memory area.
- void * chHeapAllocAligned (memory_heap_t *heapp, size_t size, unsigned align)

  Allocates a block of memory from the heap by using the first-fit algorithm.
- void chHeapFree (void *p)

Frees a previously allocated memory block.

- size_t chHeapStatus (memory_heap_t *heapp, size_t *totalp, size_t *largestp)

  Reports the heap status.

- static void * chHeapAlloc (memory_heap_t *heapp, size_t size)

  Allocates a block of memory from the heap by using the first-fit algorithm.

- static size_t chHeapGetSize (const void *p)

  Returns the size of an allocated block.

Variables

- static memory_heap_t default_heap

  Default heap descriptor.

6.12.2 Macro Definition Documentation

6.12.2.1 #define CH_HEAP_ALIGNMENT 8U

Minimum alignment used for heap.

Note

Cannot use the sizeof operator in this macro.

6.12.2.2 #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.

6.12.3 Typedef Documentation

6.12.3.1 typedef struct memory_heap memory_heap_t

Type of a memory heap.

6.12.3.2 typedef union heap_header heap_header_t

Type of a memory heap header.

6.12.4 Function Documentation

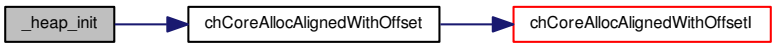
6.12.4.1 void _heap_init (void)

Initializes the default heap.

Function Class:

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

Here is the call graph for this function:



6.12.4.2 void chHeapObjectInit (memory_heap_t * *heapp*, void * *buf*, size_t *size*)

Initializes a memory heap from a static memory area.

Note

The heap buffer base and size are adjusted if the passed buffer is not aligned to `CH_HEAP_ALIGNMENT`. This mean that the effective heap size can be less than `size`.

Parameters

`out`	*heapp*	pointer to the memory heap descriptor to be initialized
`in`	*buf*	heap buffer base
`in`	*size*	heap size

Function Class:

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

6.12.4.3 void * chHeapAllocAligned (memory_heap_t * *heapp*, size_t *size*, unsigned *align*)

Allocates a block of memory from the heap by using the first-fit algorithm.

The allocated block is guaranteed to be properly aligned to the specified alignment.

Parameters

`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.

6.12.4.4 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.

6.12.4.5 size_t chHeapStatus (memory_heap_t * *heapp,* size_t * *totalp,* size_t * *largestp*)

Reports the heap status.

Note

This function is meant to be used in the test suite, it should not be really useful for the application code.

Parameters

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.

6.12.4.6 static void* chHeapAlloc (memory_heap_t * *heapp,* size_t *size*) [inline],[static]

Allocates a block of memory from the heap by using the first-fit algorithm.

The allocated block is guaranteed to be properly aligned for a pointer data type.

Parameters

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.

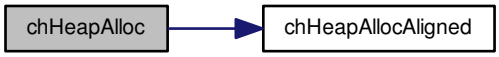
Return values

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:



6.12.4.7 static size_t chHeapGetSize (const void * *p*) [inline], [static]

Returns the size of an allocated block.

Note

The returned value is the requested size, the real size is the same value aligned to the next `CH_HEAP_AL↩IGNMENT` 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.

6.12.5 Variable Documentation

6.12.5.1 memory_heap_t default_heap [static]

Default heap descriptor.

6.13 Memory Pools

6.13.1 Detailed Description

Memory Pools related APIs and services.

Operation mode

The Memory Pools APIs allow to allocate/free fixed size objects in **constant time** and reliably without memory fragmentation problems.
Memory Pools do not enforce any alignment constraint on the contained object however the objects must be properly aligned to contain a pointer to void.

Precondition

In order to use the memory pools APIs the `CH_CFG_USE_MEMPOOLS` option must be enabled in `chconf.h`.

Note

Compatible with RT and NIL.

Macros

- #define _MEMORYPOOL_DATA(name, size, align, provider) {NULL, size, align, provider}

  Data part of a static memory pool initializer.
- #define MEMORYPOOL_DECL(name, size, align, provider) memory_pool_t name = _MEMORYPOOL_DATA(name, size, align, provider)

  Static memory pool initializer.
- #define _GUARDEDMEMORYPOOL_DATA(name, size, align)

  Data part of a static guarded memory pool initializer.
- #define GUARDEDMEMORYPOOL_DECL(name, size, align) guarded_memory_pool_t name = _GUARDEDMEMORYPOOL_DATA(name, size, align)

  Static guarded memory pool initializer.

Data Structures

- struct pool_header

  Memory pool free object header.
- struct memory_pool_t

  Memory pool descriptor.
- struct guarded_memory_pool_t

  Guarded memory pool descriptor.

Functions

- void chPoolObjectInitAligned (memory_pool_t *mp, size_t size, unsigned align, memgetfunc_t provider)

  Initializes an empty memory pool.
- void chPoolLoadArray (memory_pool_t *mp, void *p, size_t n)

  Loads a memory pool with an array of static objects.
- void * chPoolAlloc (memory_pool_t *mp)

  Allocates an object from a memory pool.

- void * 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.

6.13.2 Macro Definition Documentation

6.13.2.1 #define _MEMORYPOOL_DATA(*name, size, align, provider*) {NULL, size, align, provider}

Data part of a static memory pool initializer.

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

Parameters

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

6.13.2.2 #define MEMORYPOOL_DECL(*name, size, align, provider*) memory_pool_t name = _MEMORYPOOL_DATA(name, size, align, provider)

Static memory pool initializer.

Statically initialized memory pools require no explicit initialization using `chPoolInit()`.

Parameters

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

6.13.2.3 #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

6.13.2.4 #define GUARDEDMEMORYPOOL_DECL(*name, size, align*) guarded_memory_pool_t name = _GUARDEDMEMORYPOOL_DATA(name, size, align)

Static guarded memory pool initializer.

Statically initialized guarded memory pools require no explicit initialization using `chGuardedPoolInit()`.

Parameters

in	*name*	the name of the guarded memory pool variable
in	*size*	size of the memory pool contained objects
in	*align*	required memory alignment

6.13.3 Function Documentation

6.13.3.1 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:

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

6.13.3.2 void chPoolLoadArray (memory_pool_t * *mp*, void * *p*, size_t *n*)

Loads a memory pool with an array of static objects.

Precondition

The memory pool must already be initialized.
The array elements must be of the right size for the specified memory pool.
The array elements size must be a multiple of the alignment requirement for the pool.

Postcondition

The memory pool contains the elements of the input array.

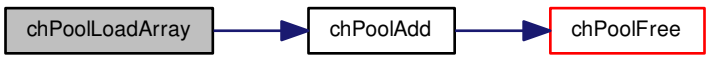
Parameters

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:



6.13.3.3 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.

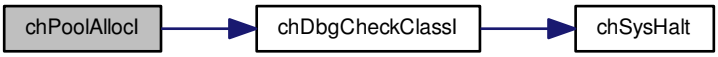
Return values

NULL	if pool is empty.

Function Class:

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

Here is the call graph for this function:



6.13.3.4 void * chPoolAlloc (memory_pool_t * *mp*)

Allocates an object from a memory pool.

Precondition

The memory pool must already be initialized.

Parameters

in	*mp*	pointer to a memory_pool_t structure

Returns

The pointer to the allocated object.

Return values

NULL	if pool is empty.

Function Class:

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

Here is the call graph for this function:

6.13.3.5 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.

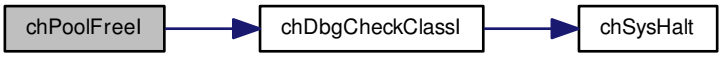
Parameters

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.

Here is the call graph for this function:



6.13.3.6 void chPoolFree (memory_pool_t * *mp*, void * *objp*)

Releases an object into a memory pool.

Precondition

The memory pool must already be initialized.
The freed object must be of the right size for the specified memory pool.
The added object must be properly aligned.

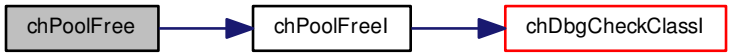
Parameters

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:



6.13.3.7 void chGuardedPoolObjectInitAligned (guarded_memory_pool_t * *gmp*, size_t *size*, unsigned *align*)

Initializes an empty guarded memory pool.

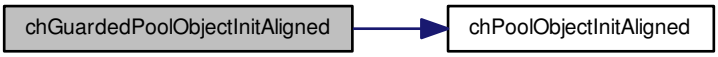
Parameters

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:

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

Here is the call graph for this function:



6.13.3.8 void chGuardedPoolLoadArray (guarded_memory_pool_t * *gmp*, void * *p*, size_t *n*)

Loads a guarded memory pool with an array of static objects.

Precondition

The guarded memory pool must already be initialized.
The array elements must be of the right size for the specified guarded memory pool.

Postcondition

The guarded memory pool contains the elements of the input array.

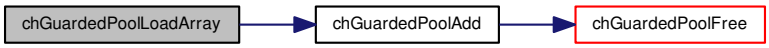
Parameters

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:



6.13.3.9 void * chGuardedPoolAllocTimeoutS (guarded_memory_pool_t * *gmp*, sysinterval_t *timeout*)

Allocates an object from a guarded memory pool.

Precondition

The guarded memory pool must already be initialized.

Parameters

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.

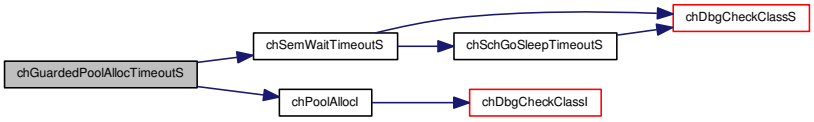
Return values

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:



6.13.3.10 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:

6.13.3.11 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.

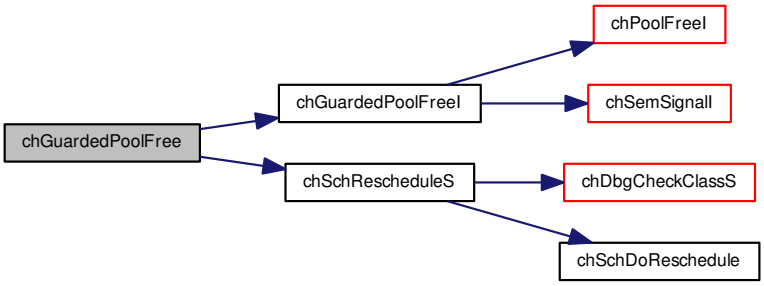
Parameters

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:



6.13.3.12 static void chPoolObjectInit (memory_pool_t * *mp*, size_t *size*, memgetfunc_t *provider*) [inline], [static]

Initializes an empty memory pool.

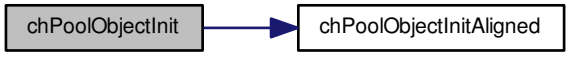
Parameters

out	*mp*	pointer to a memory_pool_t structure
in	*size*	the size of the objects contained in this memory pool, the minimum accepted size is the size of a pointer to void.
in	*provider*	memory provider function for the memory pool or NULL if the pool is not allowed to grow automatically

Function Class:

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

Here is the call graph for this function:



6.13.3.13 static void chPoolAdd (memory_pool_t * *mp*, void * *objp*) [inline], [static]

Adds an object to a memory pool.

Precondition

The memory pool must be already been initialized.
The added object must be of the right size for the specified memory pool.
The added object must be properly aligned.

Note

This function is just an alias for chPoolFree() and has been added for clarity.

Parameters

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:

6.13.3.14 static void chPoolAddI (memory_pool_t * *mp*, void * *objp*) [inline], [static]

Adds an object to a memory pool.

Precondition

The memory pool must be already been initialized.
The added object must be of the right size for the specified memory pool.
The added object must be properly aligned.

Note

This function is just an alias for `chPoolFreeI()` and has been added for clarity.

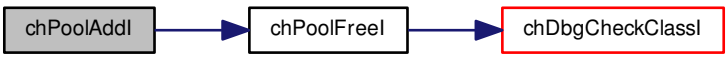
Parameters

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:



6.13.3.15 static void chGuardedPoolObjectInit (guarded_memory_pool_t * *gmp*, size_t *size*) [inline], [static]

Initializes an empty guarded memory pool.

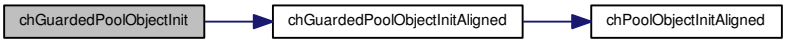
Parameters

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:

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

Here is the call graph for this function:



6.13.3.16 static cnt_t chGuardedPoolGetCounterI (guarded_memory_pool_t * *gmp*) [inline], [static]

Gets the count of objects in a guarded memory pool.

Precondition

The guarded memory pool must be already been initialized.

Parameters

in	*gmp*	pointer to a guarded_memory_pool_t structure

Returns

The number of objects.

Function Class:

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

6.13.3.17 static void* chGuardedPoolAllocI (guarded_memory_pool_t * *gmp*) [inline], [static]

Allocates an object from a guarded memory pool.

Precondition

The guarded memory pool must be already been initialized.

Parameters

in	*gmp*	pointer to a guarded_memory_pool_t structure

Returns

The pointer to the allocated object.

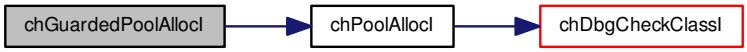
Return values

NULL	if the pool is empty.

Function Class:

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

Here is the call graph for this function:



6.13.3.18 static void chGuardedPoolFreeI (guarded_memory_pool_t * *gmp*, void * *objp*) [inline], [static]

Releases an object into a guarded memory pool.

Precondition

The guarded memory pool must already be initialized.
The freed object must be of the right size for the specified guarded memory pool.
The added object must be properly aligned.

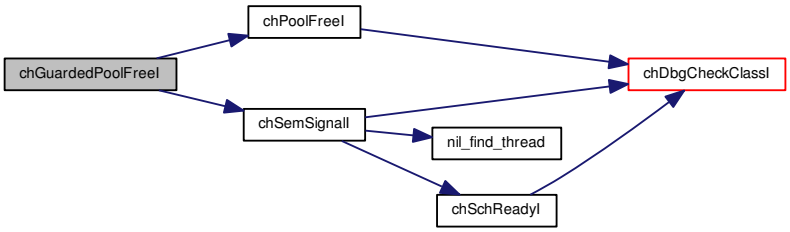
Parameters

in	*gmp*	pointer to a guarded_memory_pool_t structure
in	*objp*	the pointer to the object to be released

Function Class:

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

Here is the call graph for this function:



6.13.3.19 static void chGuardedPoolFreeS (guarded_memory_pool_t * *gmp*, void * *objp*) [inline], [static]

Releases an object into a guarded memory pool.

Precondition

The guarded memory pool must already be initialized.
The freed object must be of the right size for the specified guarded memory pool.
The added object must be properly aligned.

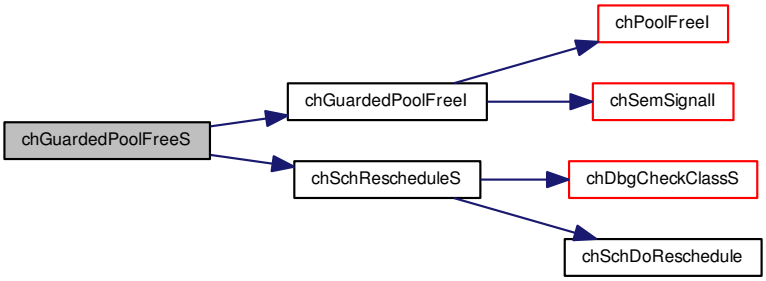
Parameters

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:



6.13.3.20 static void chGuardedPoolAdd (guarded_memory_pool_t * *gmp*, void * *objp*) [inline], [static]

Adds an object to a guarded memory pool.

Precondition

The guarded memory pool must be already been initialized.
The added object must be of the right size for the specified guarded memory pool.
The added object must be properly aligned.

Note

This function is just an alias for `chGuardedPoolFree()` and has been added for clarity.

Parameters

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:

6.13.3.21 static void chGuardedPoolAddI (guarded_memory_pool_t * *gmp*, void * *objp*) [inline], [static]

Adds an object to a guarded memory pool.

Precondition

The guarded memory pool must be already been initialized.
The added object must be of the right size for the specified guarded memory pool.
The added object must be properly aligned.

Note

This function is just an alias for `chGuardedPoolFreeI()` and has been added for clarity.

Parameters

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:

6.13.3.22 static void chGuardedPoolAddS (guarded_memory_pool_t * *gmp*, void * *objp*) [inline], [static]

Adds an object to a guarded memory pool.

Precondition

The guarded memory pool must be already been initialized.
The added object must be of the right size for the specified guarded memory pool.
The added object must be properly aligned.

Note

This function is just an alias for `chGuardedPoolFreeI()` and has been added for clarity.

Parameters

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:

6.14 Complex Services

6.14.1 Detailed Description

Modules

- Objects FIFOs
- Dynamic Objects Factory

6.15 Objects FIFOs

6.15.1 Detailed Description

Typedefs

- typedef struct ch_objects_fifo objects_fifo_t

  Type of an objects FIFO.

Data Structures

- struct ch_objects_fifo

  Type of an objects FIFO.

Functions

- static void chFifoObjectInitAligned (objects_fifo_t *ofp, size_t objsize, size_t objn, unsigned objalign, void *objbuf, msg_t *msgbuf)

  Initializes a FIFO object.
- static void chFifoObjectInit (objects_fifo_t *ofp, size_t objsize, size_t objn, void *objbuf, msg_t *msgbuf)

  Initializes a FIFO object.
- static void * chFifoTakeObjectI (objects_fifo_t *ofp)

  Allocates a free object.
- static void * chFifoTakeObjectTimeoutS (objects_fifo_t *ofp, sysinterval_t timeout)

  Allocates a free object.
- static void * chFifoTakeObjectTimeout (objects_fifo_t *ofp, sysinterval_t timeout)

  Allocates a free object.
- static void chFifoReturnObjectI (objects_fifo_t *ofp, void *objp)

  Releases a fetched object.
- static void chFifoReturnObjectS (objects_fifo_t *ofp, void *objp)

  Releases a fetched object.
- static void chFifoReturnObject (objects_fifo_t *ofp, void *objp)

  Releases a fetched object.
- static void chFifoSendObjectI (objects_fifo_t *ofp, void *objp)

  Posts an object.
- static void chFifoSendObjectS (objects_fifo_t *ofp, void *objp)

  Posts an object.
- static void chFifoSendObject (objects_fifo_t *ofp, void *objp)

  Posts an object.
- static void chFifoSendObjectAheadI (objects_fifo_t *ofp, void *objp)

  Posts an high priority object.
- static void chFifoSendObjectAheadS (objects_fifo_t *ofp, void *objp)

  Posts an high priority object.
- static void chFifoSendObjectAhead (objects_fifo_t *ofp, void *objp)

  Posts an high priority object.
- static msg_t chFifoReceiveObjectI (objects_fifo_t *ofp, void **objpp)

  Fetches an object.
- static msg_t chFifoReceiveObjectTimeoutS (objects_fifo_t *ofp, void **objpp, sysinterval_t timeout)

  Fetches an object.
- static msg_t chFifoReceiveObjectTimeout (objects_fifo_t *ofp, void **objpp, sysinterval_t timeout)

  Fetches an object.

6.15.2 Typedef Documentation

6.15.2.1 typedef struct ch_objects_fifo objects_fifo_t

Type of an objects FIFO.

6.15.3 Function Documentation

6.15.3.1 static void chFifoObjectInitAligned (objects_fifo_t * *ofp*, size_t *objsize*, size_t *objn*, unsigned *objalign*, void * *objbuf*, msg_t * *msgbuf*) [inline], [static]

Initializes a FIFO object.

Precondition

The messages size must be a multiple of the alignment requirement.

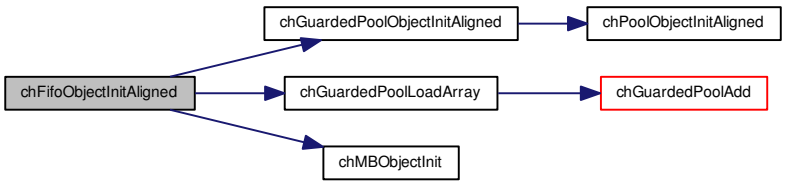
Parameters

out	*ofp*	pointer to a objects_fifo_t structure
in	*objsize*	size of objects
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 objealign alignment
in	*msgbuf*	pointer to the buffer of messages, it must be able to hold objn messages

Function Class:

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

Here is the call graph for this function:



6.15.3.2 static void chFifoObjectInit (objects_fifo_t * *ofp*, size_t *objsize*, size_t *objn*, void * *objbuf*, msg_t * *msgbuf*) [inline], [static]

Initializes a FIFO object.

Precondition

The messages size must be a multiple of the alignment requirement.

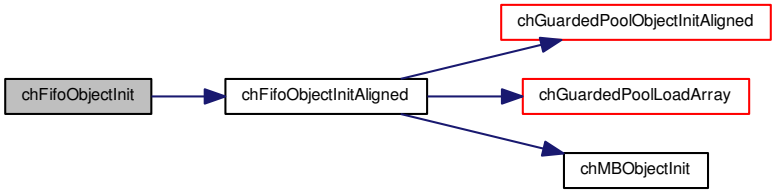
Parameters

out	*ofp*	pointer to a objects_fifo_t structure
in	*objsize*	size of objects
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 with objealign alignment
in	*msgbuf*	pointer to the buffer of messages, it must be able to hold objn messages

Function Class:

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

Here is the call graph for this function:



6.15.3.3 static void* chFifoTakeObjectI (objects_fifo_t * *ofp*) [inline], [static]

Allocates a free object.

Parameters

in	*ofp*	pointer to a objects_fifo_t structure

Returns

The pointer to the allocated object.

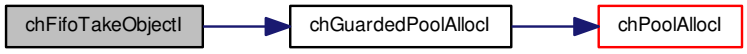
Return values

NULL	if an object is not immediately available.

Function Class:

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

Here is the call graph for this function:



6.15.3.4 static 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.

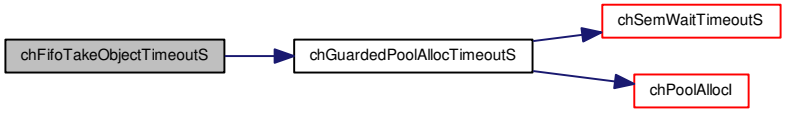
Return values

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:



6.15.3.5 static 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.

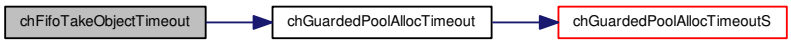
Return values

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:



6.15.3.6 static 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:

6.15.3.7 static void chFifoReturnObjectS (objects_fifo_t * *ofp*, void * *objp*) [inline], [static]

Releases a fetched object.

Parameters

in	*ofp*	pointer to a objects_fifo_t structure
in	*objp*	pointer to the object to be released

Function Class:

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

Here is the call graph for this function:

6.15.3.8 static void chFifoReturnObject (objects_fifo_t * *ofp*, void * *objp*) [inline], [static]

Releases a fetched object.

Parameters

in	*ofp*	pointer to a objects_fifo_t structure
in	*objp*	pointer to the object to be released

Function Class:

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

Here is the call graph for this function:

6.15.3.9 static void chFifoSendObjectI (objects_fifo_t * *ofp*, void * *objp*) [inline], [static]

Posts an object.

Note

By design the object can be always immediately posted.

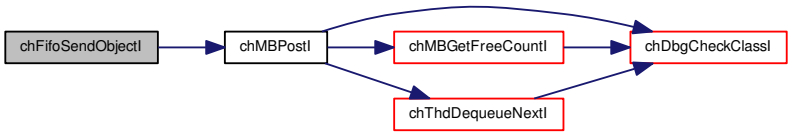
Parameters

in	*ofp*	pointer to a objects_fifo_t structure
in	*objp*	pointer to the object to be posted

Function Class:

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

Here is the call graph for this function:



6.15.3.10 static void chFifoSendObjectS (objects_fifo_t * *ofp*, void * *objp*) [inline], [static]

Posts an object.

Note

By design the object can be always immediately posted.

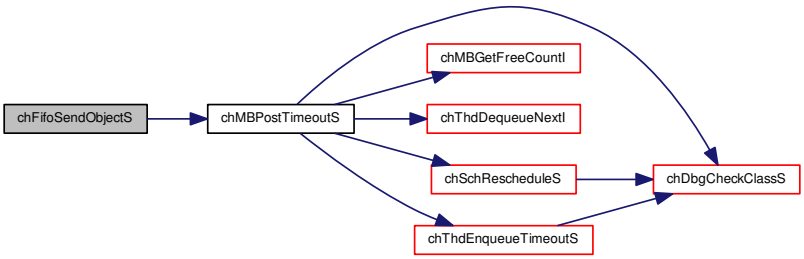
Parameters

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:



6.15.3.11 static void chFifoSendObject (objects_fifo_t * *ofp*, void * *objp*) [inline], [static]

Posts an object.

Note

By design the object can be always immediately posted.

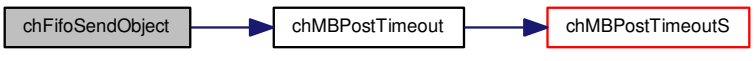
Parameters

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:



6.15.3.12 static void chFifoSendObjectAheadI (objects_fifo_t * *ofp*, void * *objp*) [inline], [static]

Posts an high priority object.

Note

By design the object can be always immediately posted.

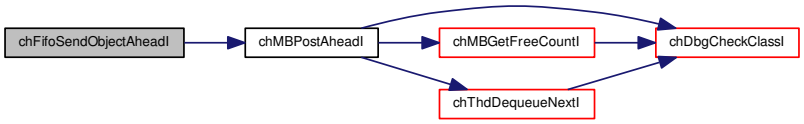
Parameters

in	*ofp*	pointer to a objects_fifo_t structure
in	*objp*	pointer to the object to be posted

Function Class:

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

Here is the call graph for this function:



6.15.3.13 static void chFifoSendObjectAheadS (objects_fifo_t * *ofp*, void * *objp*) [inline], [static]

Posts an high priority object.

Note

By design the object can be always immediately posted.

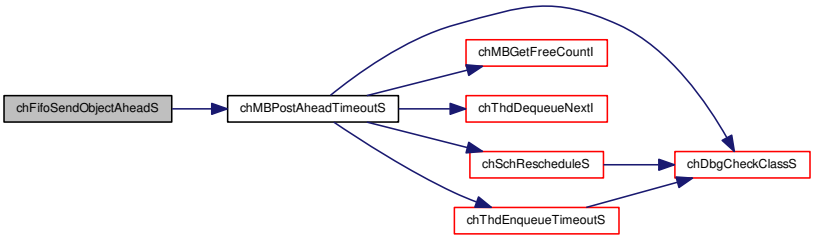
Parameters

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:



6.15.3.14 static void chFifoSendObjectAhead (objects_fifo_t * *ofp*, void * *objp*) [inline], [static]

Posts an high priority object.

Note

By design the object can be always immediately posted.

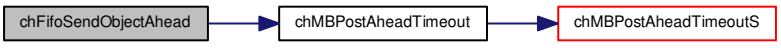
Parameters

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:



6.15.3.15 static 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.

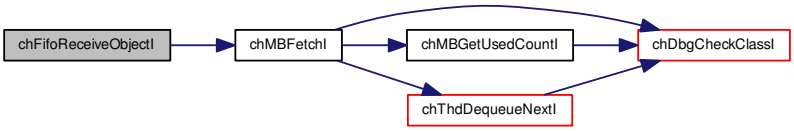
Return values

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:



6.15.3.16 static 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.

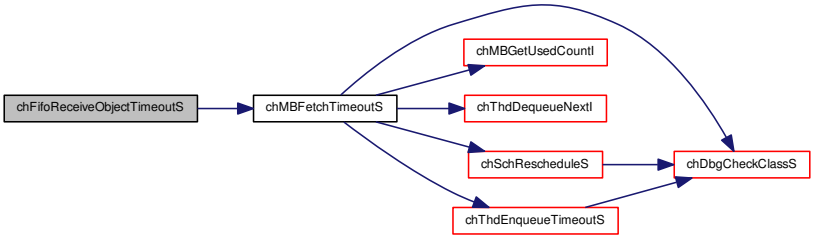
Return values

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:



6.15.3.17 static msg_t chFifoReceiveObjectTimeout (objects_fifo_t * *ofp*, void ** *objpp*, sysinterval_t *timeout*) [inline], [static]

Fetches an object.

Parameters

in	*ofp*	pointer to a `objects_fifo_t` structure
in	*objpp*	pointer to the fetched object reference
in	*timeout*	the number of ticks before the operation timeouts, the following special values are allowed: • *TIME_IMMEDIATE* immediate timeout. • *TIME_INFINITE* no timeout.

Returns

The operation status.

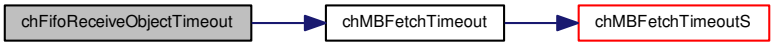
Return values

MSG_OK	if an object has been correctly fetched.
MSG_TIMEOUT	if the operation has timed out.

Function Class:

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

Here is the call graph for this function:



6.16 Dynamic Objects Factory

6.16.1 Detailed Description

The object factory is a subsystem that allows to:

- Register static objects by name.
- Dynamically create objects and assign them a name.
- Retrieve existing objects by name.
- Free objects by reference.

Allocated OS objects are handled using a reference counter, only when all references have been released then the object memory is freed in a pool.

Precondition

This subsystem requires the `CH_CFG_USE_MEMCORE` and `CH_CFG_USE_MEMPOOLS` options to be set to `TRUE`. The option `CH_CFG_USE_HEAP` is also required if the support for variable length objects is enabled.

Note

Compatible with RT and NIL.

Macros

- #define CH_CFG_FACTORY_MAX_NAMES_LENGTH 8

  Maximum length for object names.
- #define CH_CFG_FACTORY_OBJECTS_REGISTRY TRUE

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

  Enables factory for generic buffers.
- #define CH_CFG_FACTORY_SEMAPHORES TRUE

  Enables factory for semaphores.
- #define CH_CFG_FACTORY_SEMAPHORES FALSE

  Enables factory for semaphores.
- #define CH_CFG_FACTORY_MAILBOXES TRUE

  Enables factory for mailboxes.
- #define CH_CFG_FACTORY_MAILBOXES FALSE

  Enables factory for mailboxes.
- #define CH_CFG_FACTORY_OBJ_FIFOS TRUE

  Enables factory for objects FIFOs.
- #define CH_CFG_FACTORY_OBJ_FIFOS TRUE

  Enables factory for objects FIFOs.
- #define CH_CFG_FACTORY_OBJ_FIFOS FALSE

  Enables factory for objects FIFOs.
- #define CH_CFG_FACTORY_PIPES TRUE

  Enables factory for Pipes.
- #define CH_CFG_FACTORY_PIPES FALSE

  Enables factory for Pipes.

Typedefs

- typedef struct ch_dyn_element dyn_element_t

  Type of a dynamic object list element.
- typedef struct ch_dyn_list dyn_list_t

  Type of a dynamic object list.
- typedef struct ch_registered_static_object registered_object_t

  Type of a registered object.
- typedef struct ch_dyn_object dyn_buffer_t

  Type of a dynamic buffer object.
- typedef struct ch_dyn_semaphore dyn_semaphore_t

  Type of a dynamic semaphore.
- typedef struct ch_dyn_mailbox dyn_mailbox_t

  Type of a dynamic buffer object.
- typedef struct ch_dyn_objects_fifo dyn_objects_fifo_t

  Type of a dynamic buffer object.
- typedef struct ch_dyn_pipe dyn_pipe_t

  Type of a dynamic pipe object.
- typedef struct ch_objects_factory objects_factory_t

  Type of the factory main object.

Data Structures

- struct ch_dyn_element

  Type of a dynamic object list element.
- struct ch_dyn_list

  Type of a dynamic object list.
- struct ch_registered_static_object

  Type of a registered object.
- struct ch_dyn_object

  Type of a dynamic buffer object.
- struct ch_dyn_semaphore

  Type of a dynamic semaphore.
- struct ch_dyn_mailbox

  Type of a dynamic buffer object.
- struct ch_dyn_objects_fifo

  Type of a dynamic buffer object.
- struct ch_dyn_pipe

  Type of a dynamic pipe object.
- struct ch_objects_factory

  Type of the factory main object.

Functions

- void _factory_init (void)

  Initializes the objects factory.
- registered_object_t * chFactoryRegisterObject (const char *name, void *objp)

  Registers a generic object.
- registered_object_t * chFactoryFindObject (const char *name)

  Retrieves a registered object.

- registered_object_t * chFactoryFindObjectByPointer (void *objp)

  Retrieves a registered object by pointer.
- void chFactoryReleaseObject (registered_object_t *rop)

  Releases a registered object.
- dyn_buffer_t * chFactoryCreateBuffer (const char *name, size_t size)

  Creates a generic dynamic buffer object.
- dyn_buffer_t * chFactoryFindBuffer (const char *name)

  Retrieves a dynamic buffer object.
- void chFactoryReleaseBuffer (dyn_buffer_t *dbp)

  Releases a dynamic buffer object.
- dyn_semaphore_t * chFactoryCreateSemaphore (const char *name, cnt_t n)

  Creates a dynamic semaphore object.
- dyn_semaphore_t * chFactoryFindSemaphore (const char *name)

  Retrieves a dynamic semaphore object.
- void chFactoryReleaseSemaphore (dyn_semaphore_t *dsp)

  Releases a dynamic semaphore object.
- dyn_mailbox_t * chFactoryCreateMailbox (const char *name, size_t n)

  Creates a dynamic mailbox object.
- dyn_mailbox_t * chFactoryFindMailbox (const char *name)

  Retrieves a dynamic mailbox object.
- void chFactoryReleaseMailbox (dyn_mailbox_t *dmp)

  Releases a dynamic mailbox object.
- dyn_objects_fifo_t * chFactoryCreateObjectsFIFO (const char *name, size_t objsize, size_t objn, unsigned objalign)

  Creates a dynamic "objects FIFO" object.
- dyn_objects_fifo_t * 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.

6.16.2 Macro Definition Documentation

6.16.2.1 #define CH_CFG_FACTORY_MAX_NAMES_LENGTH 8

Maximum length for object names.

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

6.16.2.2 #define CH_CFG_FACTORY_OBJECTS_REGISTRY TRUE

Enables the registry of generic objects.

6.16.2.3 #define CH_CFG_FACTORY_GENERIC_BUFFERS TRUE

Enables factory for generic buffers.

6.16.2.4 #define CH_CFG_FACTORY_SEMAPHORES TRUE

Enables factory for semaphores.

6.16.2.5 #define CH_CFG_FACTORY_SEMAPHORES FALSE

Enables factory for semaphores.

6.16.2.6 #define CH_CFG_FACTORY_MAILBOXES TRUE

Enables factory for mailboxes.

6.16.2.7 #define CH_CFG_FACTORY_MAILBOXES FALSE

Enables factory for mailboxes.

6.16.2.8 #define CH_CFG_FACTORY_OBJ_FIFOS TRUE

Enables factory for objects FIFOs.

6.16.2.9 #define CH_CFG_FACTORY_OBJ_FIFOS TRUE

Enables factory for objects FIFOs.

6.16.2.10 #define CH_CFG_FACTORY_OBJ_FIFOS FALSE

Enables factory for objects FIFOs.

6.16.2.11 #define CH_CFG_FACTORY_PIPES TRUE

Enables factory for Pipes.

6.16.2.12 #define CH_CFG_FACTORY_PIPES FALSE

Enables factory for Pipes.

6.16.3 Typedef Documentation

6.16.3.1 typedef struct ch_dyn_element dyn_element_t

Type of a dynamic object list element.

6.16.3.2 typedef struct ch_dyn_list dyn_list_t

Type of a dynamic object list.

6.16.3.3 typedef struct ch_registered_static_object registered_object_t

Type of a registered object.

6.16.3.4 typedef struct ch_dyn_object dyn_buffer_t

Type of a dynamic buffer object.

6.16.3.5 typedef struct ch_dyn_semaphore dyn_semaphore_t

Type of a dynamic semaphore.

6.16.3.6 typedef struct ch_dyn_mailbox dyn_mailbox_t

Type of a dynamic buffer object.

6.16.3.7 typedef struct ch_dyn_objects_fifo dyn_objects_fifo_t

Type of a dynamic buffer object.

6.16.3.8 typedef struct ch_dyn_pipe dyn_pipe_t

Type of a dynamic pipe object.

6.16.3.9 typedef struct ch_objects_factory objects_factory_t

Type of the factory main object.

6.16.4 Function Documentation

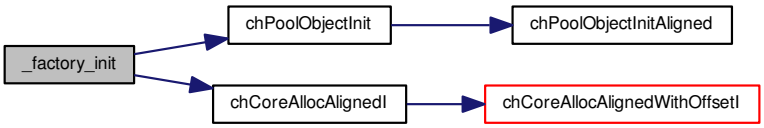
6.16.4.1 void _factory_init (void)

Initializes the objects factory.

Function Class:

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

Here is the call graph for this function:



6.16.4.2 registered_object_t * chFactoryRegisterObject (const char * *name*, void * *objp*)

Registers a generic object.

Postcondition

A reference to the registered object is returned and the reference counter is initialized to one.

Parameters

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.

6.16.4.3 registered_object_t * chFactoryFindObject (const char * *name*)

Retrieves a registered object.

Postcondition

A reference to the registered object is returned with the reference counter increased by one.

Parameters

in	*name*	name of the registered object

Returns

The reference to the found registered object.

Return values

NULL	if a registered object with the specified name does not exist.

Function Class:

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

6.16.4.4 registered_object_t * chFactoryFindObjectByPointer (void * *objp*)

Retrieves a registered object by pointer.

Postcondition

A reference to the registered object is returned with the reference counter increased by one.

Parameters

in	*objp*	pointer to the object to be retrieved

Returns

The reference to the found registered object.

Return values

NULL	if a registered object with the specified pointer does not exist.

Function Class:

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

6.16.4.5 void chFactoryReleaseObject (registered_object_t * *rop*)

Releases a registered object.

The reference counter of the registered object is decreased by one, if reaches zero then the registered object memory is freed.

Note

The object itself is not freed, it could be static, only the allocated list element is freed.

Parameters

in	*rop*	registered object reference

Function Class:

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

6.16.4.6 dyn_buffer_t * chFactoryCreateBuffer (const char * *name*, size_t *size*)

Creates a generic dynamic buffer object.

Postcondition

A reference to the dynamic buffer object is returned and the reference counter is initialized to one.
The dynamic buffer object is filled with zeros.

Parameters

in	*name*	name to be assigned to the new dynamic buffer object
in	*size*	payload size of the dynamic buffer object to be created

Returns

The reference to the created dynamic buffer object.

Return values

NULL	if the dynamic buffer object cannot be allocated or a dynamic buffer object with the same name exists.

Function Class:

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

6.16.4.7 dyn_buffer_t * chFactoryFindBuffer (const char * *name*)

Retrieves a dynamic buffer object.

Postcondition

A reference to the dynamic buffer object is returned with the reference counter increased by one.

Parameters

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.

6.16.4.8 void chFactoryReleaseBuffer (dyn_buffer_t * *dbp*)

Releases a dynamic buffer object.

The reference counter of the dynamic buffer object is decreased by one, if reaches zero then the dynamic buffer object memory is freed.

Parameters

in	*dbp*	dynamic buffer object reference

Function Class:

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

6.16.4.9 dyn_semaphore_t * chFactoryCreateSemaphore (const char * *name*, cnt_t *n*)

Creates a dynamic semaphore object.

Postcondition

A reference to the dynamic semaphore object is returned and the reference counter is initialized to one. The dynamic semaphore object is initialized and ready to use.

Parameters

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.

6.16.4.10 dyn_semaphore_t * chFactoryFindSemaphore (const char * *name*)

Retrieves a dynamic semaphore object.

Postcondition

A reference to the dynamic semaphore object is returned with the reference counter increased by one.

Parameters

in	*name*	name of the dynamic semaphore object

Returns

The reference to the found dynamic semaphore object.

Return values

NULL	if a dynamic semaphore object with the specified name does not exist.

Function Class:

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

6.16.4.11 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.

6.16.4.12 dyn_mailbox_t * chFactoryCreateMailbox (const char * *name,* size_t *n*)

Creates a dynamic mailbox object.

Postcondition

A reference to the dynamic mailbox object is returned and the reference counter is initialized to one.
The dynamic mailbox object is initialized and ready to use.

Parameters

in	*name*	name to be assigned to the new dynamic mailbox object
in	*n*	mailbox buffer size as number of messages

Returns

The reference to the created dynamic mailbox object.

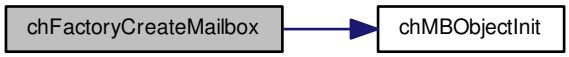
Return values

NULL	if the dynamic mailbox object cannot be allocated or a dynamic mailbox object with the same name exists.

Function Class:

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

Here is the call graph for this function:



6.16.4.13 dyn_mailbox_t * chFactoryFindMailbox (const char * *name*)

Retrieves a dynamic mailbox object.

Postcondition

A reference to the dynamic mailbox object is returned with the reference counter increased by one.

Parameters

in	*name*	name of the dynamic mailbox object

Returns

The reference to the found dynamic mailbox object.

Return values

NULL	if a dynamic mailbox object with the specified name does not exist.

Function Class:

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

6.16.4.14 void chFactoryReleaseMailbox (dyn_mailbox_t * *dmp*)

Releases a dynamic mailbox object.

The reference counter of the dynamic mailbox object is decreased by one, if reaches zero then the dynamic mailbox object memory is freed.

Parameters

in	*dmp*	dynamic mailbox object reference

Function Class:

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

6.16.4.15 dyn_objects_fifo_t * chFactoryCreateObjectsFIFO (const char * *name*, size_t *objsize*, size_t *objn*, unsigned *objalign*)

Creates a dynamic "objects FIFO" object.

Postcondition

A reference to the dynamic "objects FIFO" object is returned and the reference counter is initialized to one. The dynamic "objects FIFO" object is initialized and ready to use.

Parameters

in	*name*	name to be assigned to the new dynamic "objects FIFO" object
in	*objsize*	size of objects
in	*objn*	number of objects available
in	*objalign*	required objects alignment

Returns

The reference to the created dynamic "objects FIFO" object.

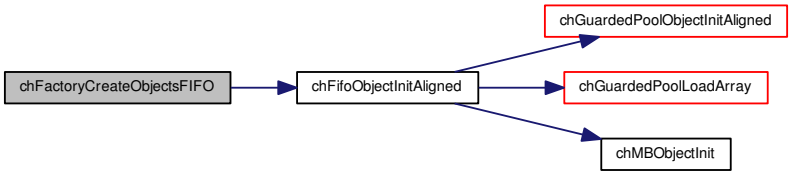
Return values

NULL	if the dynamic "objects FIFO" object cannot be allocated or a dynamic "objects FIFO" object with the same name exists.

Function Class:

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

Here is the call graph for this function:



6.16.4.16 dyn_objects_fifo_t * chFactoryFindObjectsFIFO (const char * *name*)

Retrieves a dynamic "objects FIFO" object.

Postcondition

A reference to the dynamic "objects FIFO" object is returned with the reference counter increased by one.

Parameters

in	*name*	name of the dynamic "objects FIFO" object

Returns

The reference to the found dynamic "objects FIFO" object.

Return values

NULL	if a dynamic "objects FIFO" object with the specified name does not exist.

Function Class:

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

6.16.4.17 void chFactoryReleaseObjectsFIFO (dyn_objects_fifo_t * *dofp*)

Releases a dynamic "objects FIFO" object.

The reference counter of the dynamic "objects FIFO" object is decreased by one, if reaches zero then the dynamic "objects FIFO" object memory is freed.

Parameters

in	*dofp*	dynamic "objects FIFO" object reference

Function Class:

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

6.16.4.18 dyn_pipe_t * chFactoryCreatePipe (const char * *name*, size_t *size*)

Creates a dynamic pipe object.

Postcondition

A reference to the dynamic pipe object is returned and the reference counter is initialized to one.
The dynamic pipe object is initialized and ready to use.

Parameters

in	*name*	name to be assigned to the new dynamic pipe object
in	*size*	pipe buffer size

Returns

The reference to the created dynamic pipe object.

Return values

NULL	if the dynamic pipe object cannot be allocated or a dynamic pipe object with the same name exists.

Function Class:

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

Here is the call graph for this function:

6.16.4.19 dyn_pipe_t * chFactoryFindPipe (const char * *name*)

Retrieves a dynamic pipe object.

Postcondition

A reference to the dynamic pipe object is returned with the reference counter increased by one.

Parameters

in	*name*	name of the pipe object

Returns

The reference to the found dynamic pipe object.

Return values

NULL	if a dynamic pipe object with the specified name does not exist.

Function Class:

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

6.16.4.20 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.

6.16.4.21 static dyn_element_t* chFactoryDuplicateReference (dyn_element_t * *dep*) [inline], [static]

Duplicates an object reference.

Note

This function can be used on any kind of dynamic object.

Parameters

in	*dep*	pointer to the element field of the object

Returns

The duplicated object reference.

Function Class:

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

6.16.4.22 static void* chFactoryGetObject (registered_object_t * *rop*) [inline], [static]

Returns the pointer to the inner registered object.

Parameters

in	*rop*	registered object reference

Returns

The pointer to the registered object.

Function Class:

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

6.16.4.23 static size_t chFactoryGetBufferSize (dyn_buffer_t * *dbp*) [inline], [static]

Returns the size of a generic dynamic buffer object.

Parameters

in	*dbp*	dynamic buffer object reference

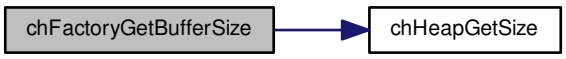
Returns

The size of the buffer object in bytes.

Function Class:

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

Here is the call graph for this function:



6.16.4.24 static uint8_t* chFactoryGetBuffer (dyn_buffer_t * *dbp*) [inline], [static]

Returns the pointer to the inner buffer.

Parameters

in	*dbp*	dynamic buffer object reference

Returns

The pointer to the dynamic buffer.

Function Class:

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

6.16.4.25 static semaphore_t* chFactoryGetSemaphore (dyn_semaphore_t * *dsp*) [inline], [static]

Returns the pointer to the inner semaphore.

Parameters

in	*dsp*	dynamic semaphore object reference

Returns

The pointer to the semaphore.

Function Class:

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

6.16.4.26 static mailbox_t* chFactoryGetMailbox (dyn_mailbox_t * *dmp*) [inline], [static]

Returns the pointer to the inner mailbox.

Parameters

in	*dmp*	dynamic mailbox object reference

Returns

The pointer to the mailbox.

Function Class:

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

6.16.4.27 static objects_fifo_t* chFactoryGetObjectsFIFO (dyn_objects_fifo_t * *dofp*) [inline], [static]

Returns the pointer to the inner objects FIFO.

Parameters

in	*dofp*	dynamic "objects FIFO" object reference

Returns

The pointer to the objects FIFO.

Function Class:

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

6.16.4.28 static pipe_t* chFactoryGetPipe (dyn_pipe_t * *dpp*) [inline], [static]

Returns the pointer to the inner pipe.

Parameters

in	*dpp*	dynamic pipe object reference

Returns

The pointer to the pipe.

Function Class:

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

6.16.5 Variable Documentation

6.16.5.1 objects_factory_t ch_factory

Factory object static instance.

Note

It is a global object because it could be accessed through a specific debugger plugin.

Chapter 7

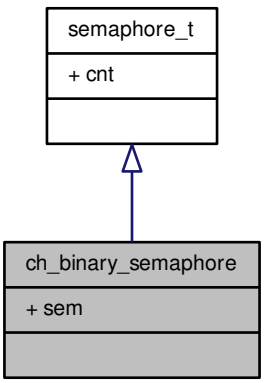
Data Structure Documentation

7.1 ch_binary_semaphore Struct Reference

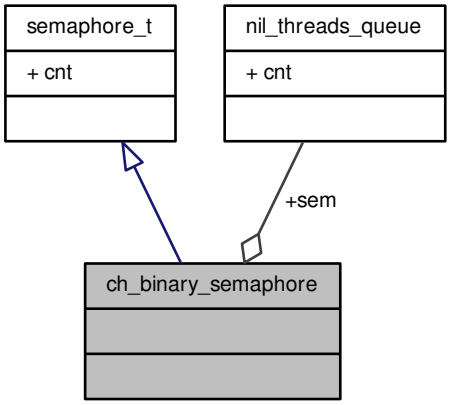
Binary semaphore type.

`#include <chbsem.h>`

Inheritance diagram for ch_binary_semaphore:



Collaboration diagram for ch_binary_semaphore:



Additional Inherited Members

7.1.1 Detailed Description

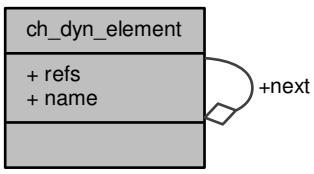
Binary semaphore type.

7.2 ch_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.

7.2.1 Detailed Description

Type of a dynamic object list element.

7.2.2 Field Documentation

7.2.2.1 struct ch_dyn_element* ch_dyn_element::next

Next dynamic object in the list.

7.2.2.2 ucnt_t ch_dyn_element::refs

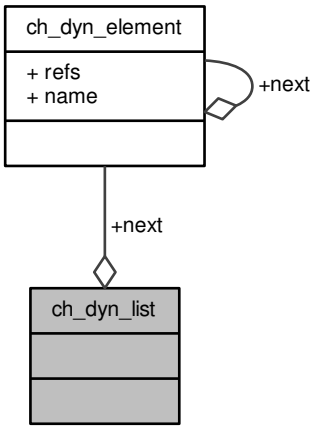
Number of references to this object.

7.3 ch_dyn_list Struct Reference

Type of a dynamic object list.

`#include <chfactory.h>`

Collaboration diagram for ch_dyn_list:



7.3.1 Detailed Description

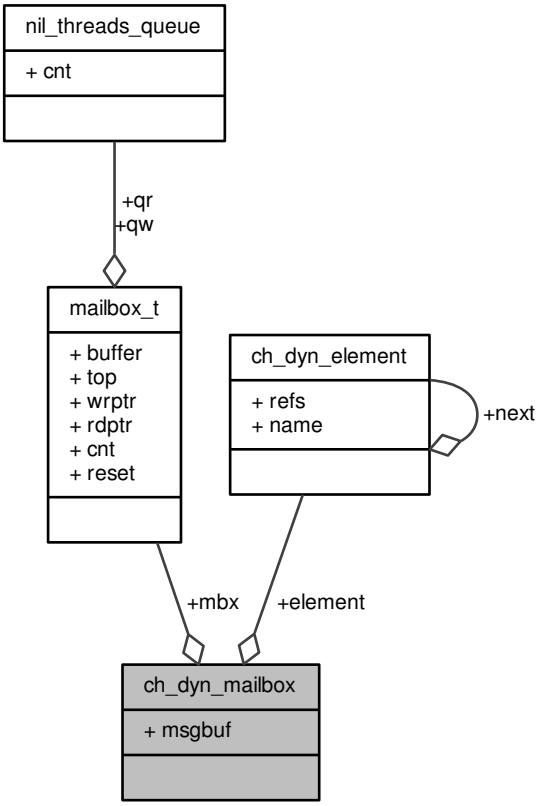
Type of a dynamic object list.

7.4 ch_dyn_mailbox Struct Reference

Type of a dynamic buffer object.

`#include <chfactory.h>`

Collaboration diagram for ch_dyn_mailbox:



Data Fields

- dyn_element_t element

  List element of the dynamic buffer object.

- mailbox_t mbx

  The mailbox.

- msg_t msgbuf [ ]

  Messages buffer.

7.4.1 Detailed Description

Type of a dynamic buffer object.

7.4.2 Field Documentation

7.4.2.1 dyn_element_t ch_dyn_mailbox::element

List element of the dynamic buffer object.

7.4.2.2 mailbox_t ch_dyn_mailbox::mbx

The mailbox.

7.4.2.3 msg_t ch_dyn_mailbox::msgbuf[ ]

Messages buffer.

Note

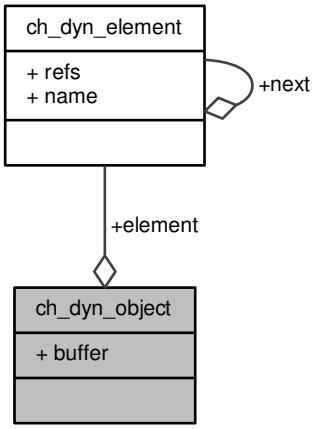
This requires C99.

7.5 ch_dyn_object Struct Reference

Type of a dynamic buffer object.

`#include <chfactory.h>`

Collaboration diagram for ch_dyn_object:



Data Fields

- dyn_element_t element

  List element of the dynamic buffer object.
- uint8_t buffer [ ]

  The buffer.

7.5.1 Detailed Description

Type of a dynamic buffer object.

7.5.2 Field Documentation

7.5.2.1 dyn_element_t ch_dyn_object::element

List element of the dynamic buffer object.

7.5.2.2 uint8_t ch_dyn_object::buffer[ ]

The buffer.

Note

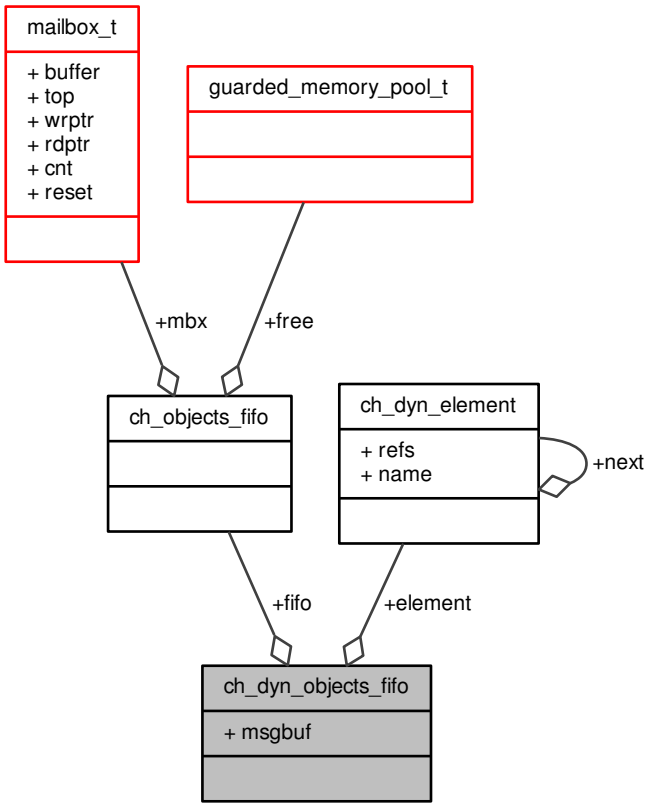
This requires C99.

7.6 ch_dyn_objects_fifo Struct Reference

Type of a dynamic buffer object.

`#include <chfactory.h>`

Collaboration diagram for ch_dyn_objects_fifo:



Data Fields

- dyn_element_t element

  List element of the dynamic buffer object.
- objects_fifo_t fifo

  The objects FIFO.
- msg_t msgbuf [ ]

  Messages buffer.

7.6.1 Detailed Description

Type of a dynamic buffer object.

7.6.2 Field Documentation

7.6.2.1 dyn_element_t ch_dyn_objects_fifo::element

List element of the dynamic buffer object.

7.6.2.2 objects_fifo_t ch_dyn_objects_fifo::fifo

The objects FIFO.

7.6.2.3 msg_t ch_dyn_objects_fifo::msgbuf[ ]

Messages buffer.

Note

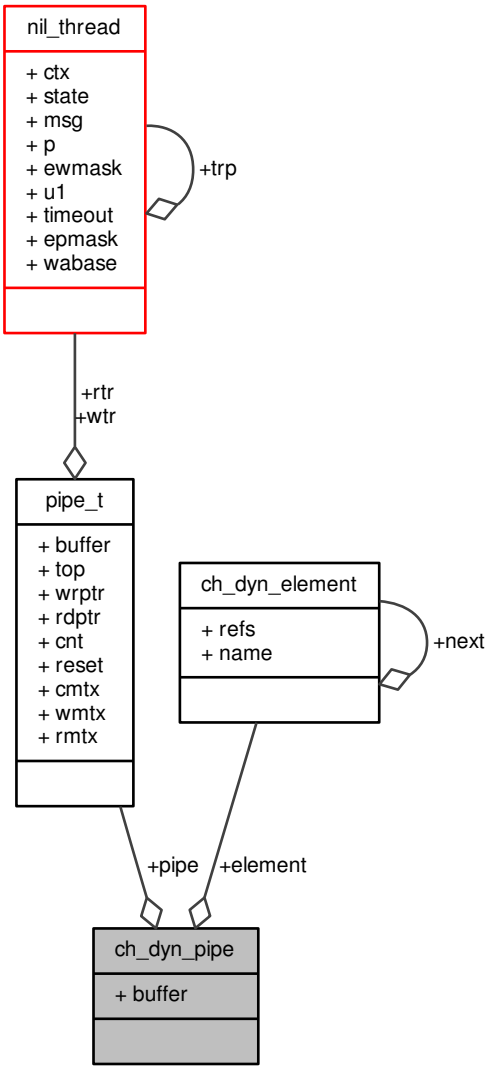
This open array is followed by another area containing the objects, this area is not represented in this structure. This requires C99.

7.7 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.
- uint8_t buffer [ ]

  Messages buffer.

7.7.1 Detailed Description

Type of a dynamic pipe object.

7.7.2 Field Documentation

7.7.2.1 dyn_element_t ch_dyn_pipe::element

List element of the dynamic pipe object.

7.7.2.2 pipe_t ch_dyn_pipe::pipe

The pipe.

7.7.2.3 uint8_t ch_dyn_pipe::buffer[ ]

Messages buffer.

Note

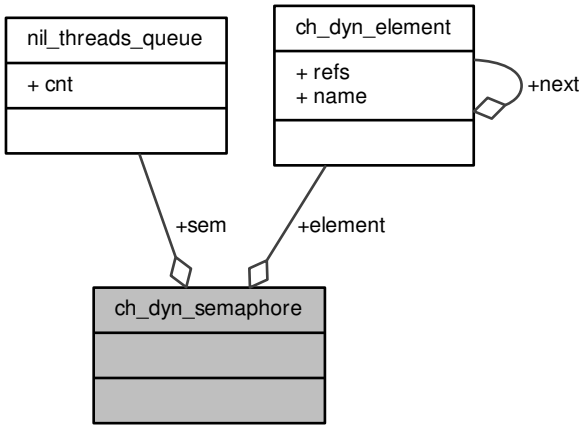
This requires C99.

7.8 ch_dyn_semaphore Struct Reference

Type of a dynamic semaphore.

`#include <chfactory.h>`

Collaboration diagram for ch_dyn_semaphore:



Data Fields

- dyn_element_t element

  List element of the dynamic semaphore.
- semaphore_t sem

  The semaphore.

7.8.1 Detailed Description

Type of a dynamic semaphore.

7.8.2 Field Documentation

7.8.2.1 dyn_element_t ch_dyn_semaphore::element

List element of the dynamic semaphore.

7.8.2.2 semaphore_t ch_dyn_semaphore::sem

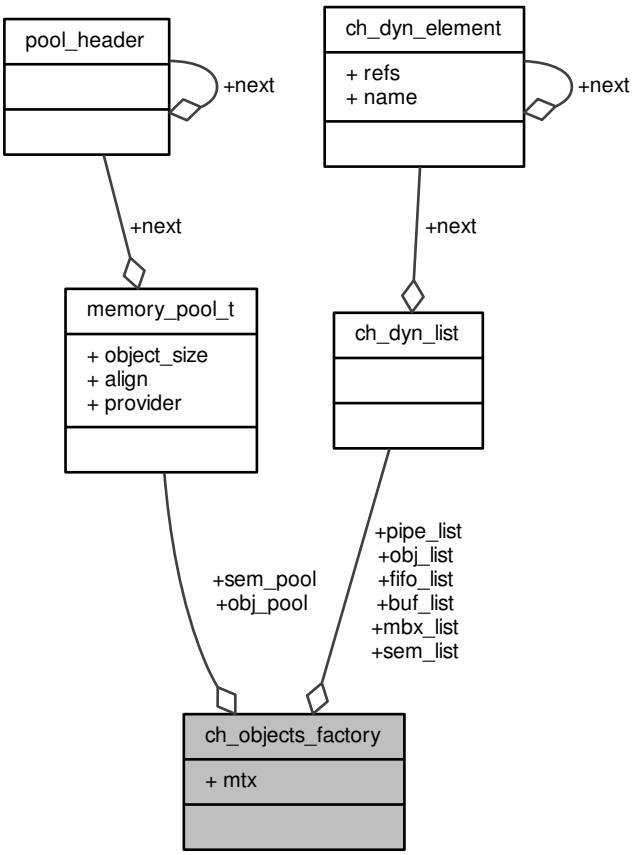
The semaphore.

7.9 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.

7.9.1 Detailed Description

Type of the factory main object.

7.9.2 Field Documentation

7.9.2.1 mutex_t ch_objects_factory::mtx

Factory access mutex or semaphore.

7.9.2.2 dyn_list_t ch_objects_factory::obj_list

List of the registered objects.

7.9.2.3 memory_pool_t ch_objects_factory::obj_pool

Pool of the available registered objects.

7.9.2.4 dyn_list_t ch_objects_factory::buf_list

List of the allocated buffer objects.

7.9.2.5 dyn_list_t ch_objects_factory::sem_list

List of the allocated semaphores.

7.9.2.6 memory_pool_t ch_objects_factory::sem_pool

Pool of the available semaphores.

7.9.2.7 dyn_list_t ch_objects_factory::mbx_list

List of the allocated buffer objects.

7.9.2.8 dyn_list_t ch_objects_factory::fifo_list

List of the allocated "objects FIFO" objects.

7.9.2.9 dyn_list_t ch_objects_factory::pipe_list

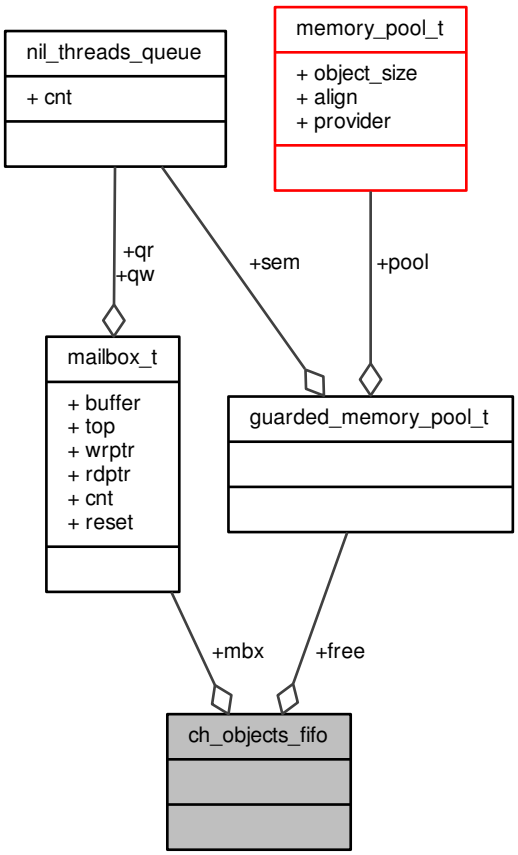
List of the allocated pipe objects.

7.10 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.

7.10.1 Detailed Description

Type of an objects FIFO.

7.10.2 Field Documentation

7.10.2.1 guarded_memory_pool_t ch_objects_fifo::free

Pool of the free objects.

7.10.2.2 mailbox_t ch_objects_fifo::mbx

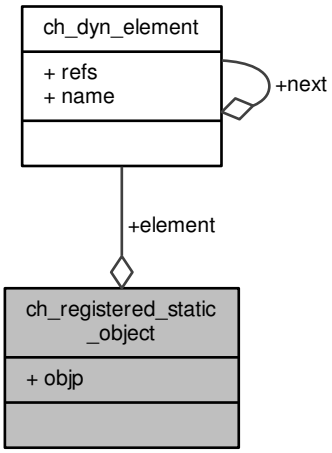
Mailbox of the sent objects.

7.11 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.

7.11.1 Detailed Description

Type of a registered object.

7.11.2 Field Documentation

7.11.2.1 dyn_element_t ch_registered_static_object::element

List element of the registered object.

7.11.2.2 void* ch_registered_static_object::objp

Pointer to the object.

Note

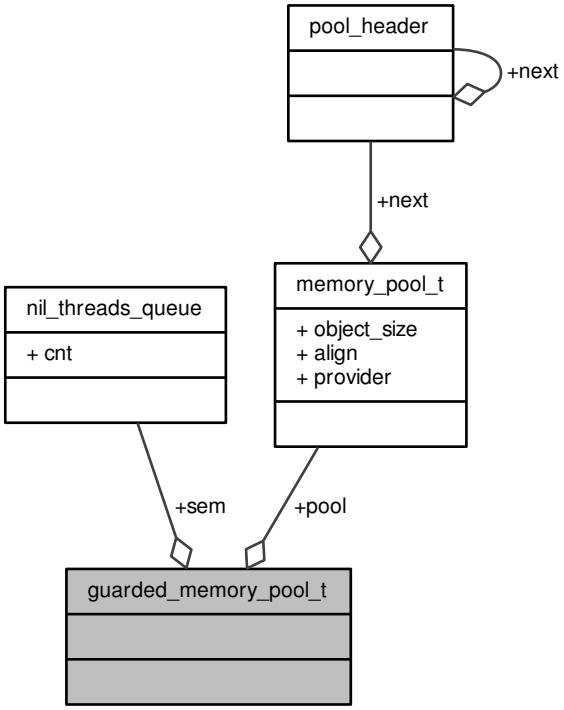
The type of the object is not stored in anyway.

7.12 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.

7.12.1 Detailed Description

Guarded memory pool descriptor.

7.12.2 Field Documentation

7.12.2.1 semaphore_t guarded_memory_pool_t::sem

Counter semaphore guarding the memory pool.

7.12.2.2 memory_pool_t guarded_memory_pool_t::pool

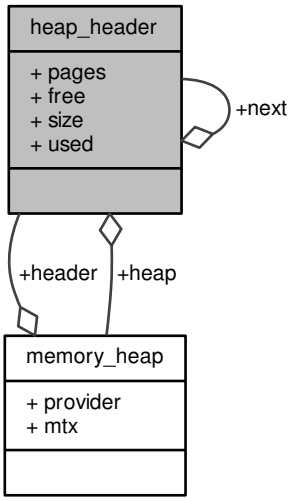
The memory pool itself.

7.13 heap_header Union Reference

Memory heap block header.

`#include <chmemheaps.h>`

Collaboration diagram for heap_header:



7.13.1 Detailed Description

Memory heap block header.

7.13.2 Field Documentation

7.13.2.1 heap_header_t* heap_header::next

Next block in free list.

7.13.2.2 size_t heap_header::pages

Size of the area in pages.

7.13.2.3 memory_heap_t* heap_header::heap

Block owner heap.

7.13.2.4 size_t heap_header::size

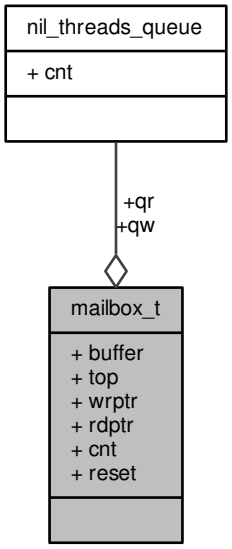
Size of the area in bytes.

7.14 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.

7.14.1 Detailed Description

Structure representing a mailbox object.

7.14.2 Field Documentation

7.14.2.1 msg_t* mailbox_t::buffer

Pointer to the mailbox buffer.

7.14.2.2 msg_t* mailbox_t::top

Pointer to the location after the buffer.

7.14.2.3 msg_t* mailbox_t::wrptr

Write pointer.

7.14.2.4 msg_t* mailbox_t::rdptr

Read pointer.

7.14.2.5 size_t mailbox_t::cnt

Messages in queue.

7.14.2.6 bool mailbox_t::reset

True in reset state.

7.14.2.7 threads_queue_t mailbox_t::qw

Queued writers.

7.14.2.8 threads_queue_t mailbox_t::qr

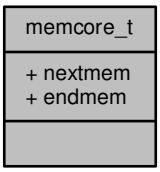
Queued readers.

7.15 memcore_t Struct Reference

Type of memory core object.

`#include <chmemcore.h>`

Collaboration diagram for memcore_t:



Data Fields

- uint8_t * nextmem

  Next free address.

- uint8_t * endmem

  Final address.

7.15.1 Detailed Description

Type of memory core object.

7.15.2 Field Documentation

7.15.2.1 uint8_t* memcore_t::nextmem

Next free address.

7.15.2.2 uint8_t* memcore_t::endmem

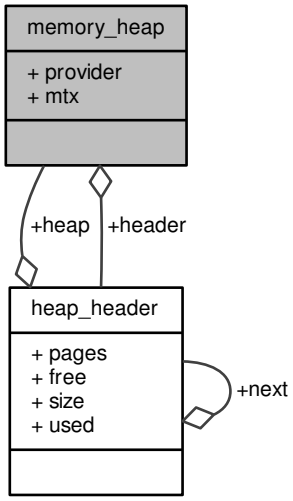
Final address.

7.16 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.

7.16.1 Detailed Description

Structure describing a memory heap.

7.16.2 Field Documentation

7.16.2.1 memgetfunc2_t memory_heap::provider

Memory blocks provider for this heap.

7.16.2.2 heap_header_t memory_heap::header

Free blocks list header.

7.16.2.3 mutex_t memory_heap::mtx

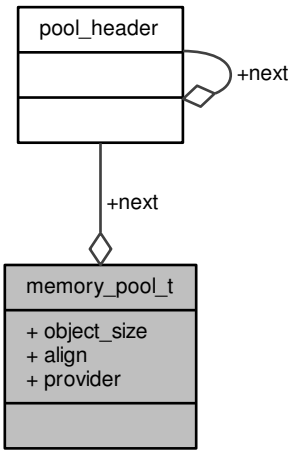
Heap access mutex.

7.17 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.

7.17.1 Detailed Description

Memory pool descriptor.

7.17.2 Field Documentation

7.17.2.1 struct pool_header* memory_pool_t::next

Pointer to the header.

7.17.2.2 size_t memory_pool_t::object_size

Memory pool objects size.

7.17.2.3 unsigned memory_pool_t::align

Required alignment.

7.17.2.4 memgetfunc_t memory_pool_t::provider

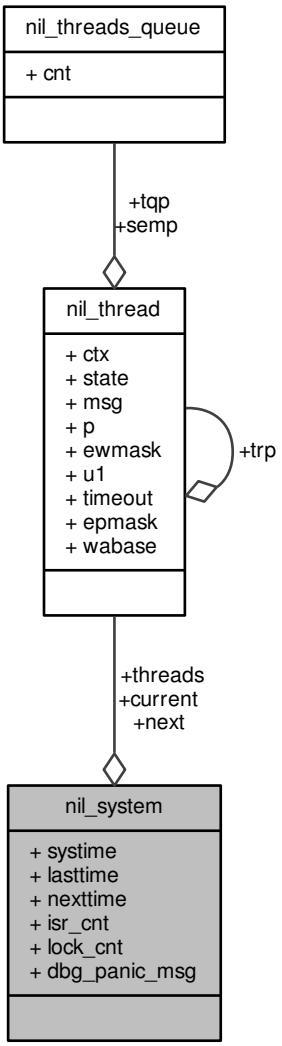
Memory blocks provider for this pool.

7.18 nil_system Struct Reference

System data structure.

`#include <ch.h>`

Collaboration diagram for nil_system:



Data Fields

- thread_t * current

  Pointer to the running thread.
- thread_t * next

  Pointer to the next thread to be executed.
- volatile systime_t systime

  System time.
- systime_t lasttime

  System time of the last tick event.
- systime_t nexttime

  Time of the next scheduled tick event.
- cnt_t isr_cnt

  ISR nesting level.
- cnt_t lock_cnt

  Lock nesting level.
- const char *volatile dbg_panic_msg

  Panic message.
- thread_t threads [CH_CFG_NUM_THREADS+1]

  Thread structures for all the defined threads.

7.18.1 Detailed Description

System data structure.

Note

This structure contain all the data areas used by the OS except stacks.

7.18.2 Field Documentation

7.18.2.1 thread_t* nil_system::current

Pointer to the running thread.

7.18.2.2 thread_t* nil_system::next

Pointer to the next thread to be executed.

Note

This pointer must point at the same thread pointed by `current` or to an higher priority thread if a switch is required.

7.18.2.3 volatile systime_t nil_system::systime

System time.

7.18.2.4 systime_t nil_system::lasttime

System time of the last tick event.

7.18.2.5 systime_t nil_system::nexttime

Time of the next scheduled tick event.

7.18.2.6 cnt_t nil_system::isr_cnt

ISR nesting level.

7.18.2.7 cnt_t nil_system::lock_cnt

Lock nesting level.

7.18.2.8 const char* volatile nil_system::dbg_panic_msg

Panic message.

Note

This field is only present if some debug options have been activated.
Accesses to this pointer must never be optimized out so the field itself is declared volatile.

7.18.2.9 thread_t nil_system::threads[CH_CFG_NUM_THREADS+1]

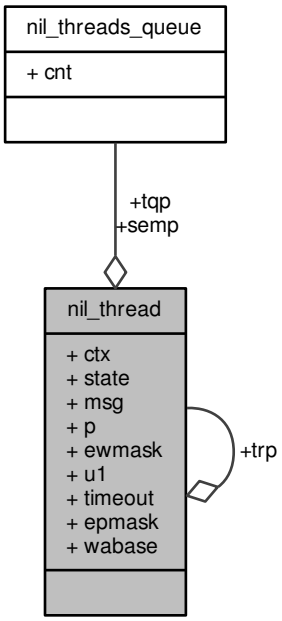
Thread structures for all the defined threads.

7.19 nil_thread Struct Reference

Structure representing a thread.

`#include <ch.h>`

Collaboration diagram for nil_thread:



Data Fields

- struct port_context ctx

  Processor context.
- tstate_t state

  Thread state.
- volatile sysinterval_t timeout

  Timeout counter, zero if disabled.
- eventmask_t epmask

  Pending events mask.
- stkalign_t * wabase

  Thread stack boundary.
- msg_t msg

  Wake-up message.
- void * p

  Generic pointer.
- thread_reference_t * trp

  Pointer to thread reference.
- threads_queue_t * tqp

  Pointer to thread queue.
- semaphore_t * semp

  Pointer to semaphore.
- eventmask_t ewmask

  Enabled events mask.

7.19.1 Detailed Description

Structure representing a thread.

7.19.2 Field Documentation

7.19.2.1 struct port_context nil_thread::ctx

Processor context.

7.19.2.2 tstate_t nil_thread::state

Thread state.

7.19.2.3 msg_t nil_thread::msg

Wake-up message.

7.19.2.4 void* nil_thread::p

Generic pointer.

7.19.2.5 thread_reference_t* nil_thread::trp

Pointer to thread reference.

7.19.2.6 threads_queue_t* nil_thread::tqp

Pointer to thread queue.

7.19.2.7 semaphore_t* nil_thread::semp

Pointer to semaphore.

7.19.2.8 eventmask_t nil_thread::ewmask

Enabled events mask.

7.19.2.9 volatile sysinterval_t nil_thread::timeout

Timeout counter, zero if disabled.

7.19.2.10 eventmask_t nil_thread::epmask

Pending events mask.

7.19.2.11 stkalign_t* nil_thread::wabase

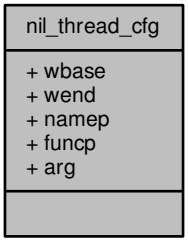
Thread stack boundary.

7.20 nil_thread_cfg Struct Reference

Structure representing a thread static configuration.

`#include <ch.h>`

Collaboration diagram for nil_thread_cfg:



Data Fields

- stkalign_t * wbase
  Thread working area base.
- stkalign_t * wend
  Thread working area end.
- const char * namep
  Thread name, for debugging.
- tfunc_t funcp
  Thread function.
- void * arg
  Thread function argument.

7.20.1 Detailed Description

Structure representing a thread static configuration.

7.20.2 Field Documentation

7.20.2.1 stkalign_t* nil_thread_cfg::wbase

Thread working area base.

7.20.2.2 stkalign_t* nil_thread_cfg::wend

Thread working area end.

7.20.2.3 const char* nil_thread_cfg::namep

Thread name, for debugging.

7.20.2.4 tfunc_t nil_thread_cfg::funcp

Thread function.

7.20.2.5 void* nil_thread_cfg::arg

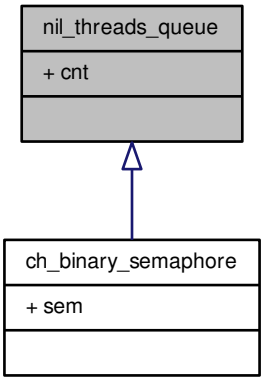
Thread function argument.

7.21 nil_threads_queue Struct Reference

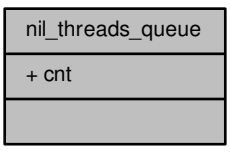
Structure representing a queue of threads.

`#include <ch.h>`

Inheritance diagram for nil_threads_queue:



Collaboration diagram for nil_threads_queue:



Data Fields

- volatile cnt_t cnt

  Threads Queue counter.

7.21.1 Detailed Description

Structure representing a queue of threads.

7.21.2 Field Documentation

7.21.2.1 volatile cnt_t nil_threads_queue::cnt

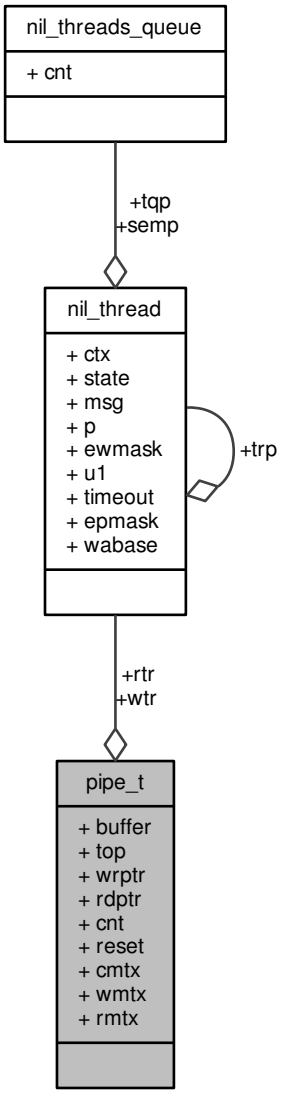
Threads Queue counter.

7.22 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.

7.22.1 Detailed Description

Structure representing a pipe object.

7.22.2 Field Documentation

7.22.2.1 uint8_t* pipe_t::buffer

Pointer to the pipe buffer.

7.22.2.2 uint8_t* pipe_t::top

Pointer to the location after the buffer.

7.22.2.3 uint8_t* pipe_t::wrptr

Write pointer.

7.22.2.4 uint8_t* pipe_t::rdptr

Read pointer.

7.22.2.5 size_t pipe_t::cnt

Bytes in the pipe.

7.22.2.6 bool pipe_t::reset

True if in reset state.

7.22.2.7 thread_reference_t pipe_t::wtr

Waiting writer.

7.22.2.8 thread_reference_t pipe_t::rtr

Waiting reader.

7.22.2.9 mutex_t pipe_t::cmtx

Common access mutex.

7.22.2.10 mutex_t pipe_t::wmtx

Write access mutex.

7.22.2.11 mutex_t pipe_t::rmtx

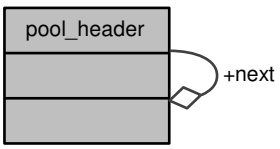
Read access mutex.

7.23 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.

7.23.1 Detailed Description

Memory pool free object header.

7.23.2 Field Documentation

7.23.2.1 struct pool_header* pool_header::next

Pointer to the next pool header in the list.

Chapter 8

File Documentation

8.1 ch.c File Reference

Nil RTOS main source file.

```
#include "ch.h"
```

Functions

- static thread_t * nil_find_thread (tstate_t state, void *p)

  Retrieves the highest priority thread in the specified state and associated to the specified object.
- static cnt_t nil_ready_all (void *p, cnt_t cnt, msg_t msg)

  Puts in ready state all thread matching the specified status and associated object.
- 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.
- void chSysInit (void)

  Initializes the kernel.

- void chSysHalt (const char *reason)

  Halts the system.
- void chSysTimerHandlerI (void)

  Time management handler.
- void chSysUnconditionalLock (void)

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

  Unconditionally leaves the kernel lock state.
- 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.
- thread_t * chSchReadyI (thread_t *tp, msg_t msg)

  Makes the specified thread ready for execution.
- bool chSchIsPreemptionRequired (void)

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

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

  Reschedules if needed.
- msg_t chSchGoSleepTimeoutS (tstate_t newstate, sysinterval_t timeout)

  Puts the current thread to sleep into the specified state with timeout specification.
- 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 chThdResume (thread_reference_t *trp, msg_t msg)

  Wakes up a thread waiting on a thread reference object.
- void chThdSleep (sysinterval_t timeout)

  Suspends the invoking thread for the specified time.
- void chThdSleepUntil (systime_t abstime)

  Suspends the invoking thread until the system time arrives to the specified value.
- msg_t chThdEnqueueTimeoutS (threads_queue_t *tqp, sysinterval_t timeout)

  Enqueues the caller thread on a threads queue object.
- void chThdDoDequeueNextI (threads_queue_t *tqp, msg_t msg)

  Dequeues and wakes up one thread from the 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.
- 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 chSemReset (semaphore_t *sp, cnt_t n)

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

  Performs a reset operation on the semaphore.
- void chEvtSignal (thread_t *tp, eventmask_t mask)

  Adds a set of event flags directly to the specified `thread_t`.
- void chEvtSignalI (thread_t *tp, eventmask_t mask)

  Adds a set of event flags directly to the specified `thread_t`.
- eventmask_t chEvtWaitAnyTimeout (eventmask_t mask, sysinterval_t timeout)

  Waits for any of the specified events.

Variables

- nil_system_t nil

  System data structures.

8.1.1 Detailed Description

Nil RTOS main source file.

8.2 ch.h File Reference

Nil RTOS main header file.

```
#include "chtypes.h"
#include "chconf.h"
#include "chlicense.h"
#include "chcore.h"
#include "chlib.h"
```

Data Structures

- struct nil_threads_queue

  Structure representing a queue of threads.
- struct nil_thread_cfg

  Structure representing a thread static configuration.
- struct nil_thread

  Structure representing a thread.
- struct nil_system

  System data structure.

Macros

- #define _CHIBIOS_NIL_

  ChibiOS/NIL identification macro.
- #define CH_KERNEL_STABLE 1

  Stable release flag.
- #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 THD_IDLE_BASE (&__main_thread_stack_base__)
- #define __CH_STRINGIFY(a) #a

  Utility to make the parameter a quoted string.

ChibiOS/NIL version identification

- #define CH_KERNEL_VERSION "3.2.2"

  Kernel version string.
- #define CH_KERNEL_MAJOR 3

  Kernel version major number.
- #define CH_KERNEL_MINOR 2

  Kernel version minor number.
- #define CH_KERNEL_PATCH 2

  Kernel version patch number.

Constants for configuration options

- #define FALSE 0

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

  Generic 'true' preprocessor boolean constant.

Wakeup messages

- #define MSG_OK (msg_t)0

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

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

  Wake-up caused by a reset condition.

Special time constants

- #define TIME_IMMEDIATE ((sysinterval_t)-1)

  Zero time specification for some functions with a timeout specification.
- #define TIME_INFINITE ((sysinterval_t)0)

  Infinite time 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.

Thread state related macros

- #define NIL_STATE_READY (tstate_t)0
  Thread ready or executing.
- #define NIL_STATE_SLEEPING (tstate_t)1
  Thread sleeping.
- #define NIL_STATE_SUSP (tstate_t)2
  Thread suspended.
- #define NIL_STATE_WTQUEUE (tstate_t)3
  On queue or semaph.
- #define NIL_STATE_WTOREVT (tstate_t)4
  Waiting for events.
- #define **NIL_THD_IS_READY**(tp) ((tp)->state == NIL_STATE_READY)
- #define **NIL_THD_IS_SLEEPING**(tp) ((tp)->state == NIL_STATE_SLEEPING)
- #define **NIL_THD_IS_SUSP**(tp) ((tp)->state == NIL_STATE_SUSP)
- #define **NIL_THD_IS_WTQUEUE**(tp) ((tp)->state == NIL_STATE_WTQUEUE)
- #define **NIL_THD_IS_WTOREVT**(tp) ((tp)->state == NIL_STATE_WTOREVT)

Events related macros

- #define ALL_EVENTS ((eventmask_t)-1)
  All events allowed mask.
- #define EVENT_MASK(eid) ((eventmask_t)(1 << (eid)))
  Returns an event mask from an event identifier.

Threads tables definition macros

- #define THD_TABLE_BEGIN const thread_config_t nil_thd_configs[CH_CFG_NUM_THREADS + 1] = {
  Start of user threads table.
- #define THD_TABLE_ENTRY(wap, name, funcp, arg)
  Entry of user threads table.
- #define THD_TABLE_END
  End of user threads table.

Memory alignment support macros

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

Working Areas

- #define THD_WORKING_AREA_SIZE(n)
  Calculates the total Working Area size.
- #define THD_WORKING_AREA(s, n) PORT_WORKING_AREA(s, n)
  Static working area allocation.

Threads abstraction macros

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

ISRs abstraction macros

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

Fast ISRs abstraction macros

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

Time conversion utilities

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

Threads queues

- #define _THREADS_QUEUE_DATA(name) {(cnt_t)0}
  Data part of a static threads queue object initializer.
- #define _THREADS_QUEUE_DECL(name) threads_queue_t name = _THREADS_QUEUE_DATA(name)
  Static threads queue object initializer.

Semaphores macros

- #define _SEMAPHORE_DATA(name, n) {n}
  Data part of a static semaphore initializer.
- #define SEMAPHORE_DECL(name, n) semaphore_t name = _SEMAPHORE_DATA(name, n)
  Static semaphore initializer.

Macro Functions

- #define chSysGetRealtimeCounterX() (rtcnt_t)port_rt_get_counter_value()
  Returns the current value of the system real time counter.
- #define chSysDisable()
  Raises the system interrupt priority mask to the maximum level.
- #define chSysSuspend()
  Raises the system interrupt priority mask to system level.
- #define chSysEnable()
  Lowers the system interrupt priority mask to user level.
- #define chSysLock()

Enters the kernel lock state.

- #define chSysUnlock()

  Leaves the kernel lock state.
- #define chSysLockFromISR()

  Enters the kernel lock state from within an interrupt handler.
- #define chSysUnlockFromISR()

  Leaves the kernel lock state from within an interrupt handler.
- #define chSchIsRescRequiredI() ((bool)(nil.current != nil.next))

  Evaluates if a reschedule is required.
- #define chThdGetSelfX() nil.current

  Returns a pointer to the current `thread_t`.
- #define chThdSleepSeconds(secs) chThdSleep(TIME_S2I(secs))

  Delays the invoking thread for the specified number of seconds.
- #define chThdSleepMilliseconds(msecs) chThdSleep(TIME_MS2I(msecs))

  Delays the invoking thread for the specified number of milliseconds.
- #define chThdSleepMicroseconds(usecs) chThdSleep(TIME_US2I(usecs))

  Delays the invoking thread for the specified number of microseconds.
- #define chThdSleepS(timeout) (void) chSchGoSleepTimeoutS(NIL_STATE_SLEEPING, timeout)

  Suspends the invoking thread for the specified time.
- #define chThdSleepUntilS(abstime)

  Suspends the invoking thread until the system time arrives to the specified value.
- #define chThdQueueObjectInit(tqp) ((tqp)->cnt = (cnt_t)0)

  Initializes a threads queue object.
- #define chThdQueueIsEmptyI(tqp) ((bool)(tqp->cnt >= (cnt_t)0))

  Evaluates to `true` if the specified queue is empty.
- #define chSemObjectInit(sp, n) ((sp)->cnt = (n))

  Initializes a semaphore with the specified counter value.
- #define chSemWait(sp) chSemWaitTimeout(sp, TIME_INFINITE)

  Performs a wait operation on a semaphore.
- #define chSemWaitS(sp) chSemWaitTimeoutS(sp, TIME_INFINITE)

  Performs a wait operation on a semaphore.
- #define chSemFastWaitI(sp) ((sp)->cnt--)

  Decreases the semaphore counter.
- #define chSemFastSignalI(sp) ((sp)->cnt++)

  Increases the semaphore counter.
- #define chSemGetCounterI(sp) ((sp)->cnt)

  Returns the semaphore counter current value.
- #define chVTGetSystemTimeX() (nil.systime)

  Current system time.
- #define chVTTimeElapsedSinceX(start) chTimeDiffX((start), chVTGetSystemTimeX())

  Returns the elapsed time since the specified start time.
- #define chTimeAddX(systime, interval) ((systime_t)(systime) + (systime_t)(interval))

  Adds an interval to a system time returning a system time.
- #define chTimeDiffX(start, end) ((sysinterval_t)((systime_t)((systime_t)(end) - (systime_t)(start))))

  Subtracts two system times returning an interval.
- #define chTimeIsInRangeX(time, start, end)

  Checks if the specified time is within the specified time range.
- #define chDbgCheck(c)

  Function parameters check.
- #define chDbgAssert(c, r)

  Condition assertion.

Typedefs

- typedef uint32_t systime_t

  Type of system time.
- typedef uint32_t sysinterval_t

  Type of time interval.

- typedef uint64_t time_conv_t

  Type of time conversion variable.
- typedef struct nil_thread thread_t

  Type of a structure representing a thread.
- typedef struct nil_threads_queue threads_queue_t

  Type of a queue of threads.
- typedef threads_queue_t semaphore_t

  Type of a structure representing a semaphore.
- typedef void(∗ tfunc_t) (void ∗p)

  Thread function.
- typedef struct nil_thread_cfg thread_config_t

  Type of a structure representing a thread static configuration.
- typedef thread_t ∗ thread_reference_t

  Type of a thread reference.
- typedef struct nil_system nil_system_t

  Type of a structure representing the system.

Functions

- void chSysInit (void)

  Initializes the kernel.
- void chSysHalt (const char ∗reason)

  Halts the system.
- void chSysTimerHandlerI (void)

  Time management handler.
- void chSysUnconditionalLock (void)

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

  Unconditionally leaves the kernel lock state.
- syssts_t chSysGetStatusAndLockX (void)

  Returns the execution status and enters a critical zone.
- bool chSysIsCounterWithinX (rtcnt_t cnt, rtcnt_t start, rtcnt_t end)

  Realtime window test.
- void chSysPolledDelayX (rtcnt_t cycles)

  Polled delay.
- void chSysRestoreStatusX (syssts_t sts)

  Restores the specified execution status and leaves a critical zone.
- thread_t ∗ chSchReadyI (thread_t ∗tp, msg_t msg)

  Makes the specified thread ready for execution.
- bool chSchIsPreemptionRequired (void)

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

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

  Reschedules if needed.
- msg_t chSchGoSleepTimeoutS (tstate_t newstate, sysinterval_t timeout)

  Puts the current thread to sleep into the specified state with timeout specification.
- 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 chThdResume (thread_reference_t *trp, msg_t msg)

  Wakes up a thread waiting on a thread reference object.
- void chThdSleep (sysinterval_t timeout)

  Suspends the invoking thread for the specified time.
- void chThdSleepUntil (systime_t abstime)

  Suspends the invoking thread until the system time arrives to the specified value.
- msg_t chThdEnqueueTimeoutS (threads_queue_t *tqp, sysinterval_t timeout)

  Enqueues the caller thread on a threads queue object.
- void chThdDoDequeueNextI (threads_queue_t *tqp, msg_t msg)

  Dequeues and wakes up one thread from the 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.
- 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 chSemReset (semaphore_t *sp, cnt_t n)

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

  Performs a reset operation on the semaphore.
- void chEvtSignal (thread_t *tp, eventmask_t mask)

  Adds a set of event flags directly to the specified `thread_t`.
- void chEvtSignalI (thread_t *tp, eventmask_t mask)

  Adds a set of event flags directly to the specified `thread_t`.
- eventmask_t chEvtWaitAnyTimeout (eventmask_t mask, sysinterval_t timeout)

  Waits for any of the specified events.

8.2.1 Detailed Description

Nil RTOS main header file.

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

8.3 chbsem.h File Reference

Binary semaphores structures and macros.

Data Structures

- struct ch_binary_semaphore

  Binary semaphore type.

Macros

- #define _BSEMAPHORE_DATA(name, taken) {_SEMAPHORE_DATA(name.sem, ((taken) ? 0 : 1))}

  Data part of a static semaphore initializer.
- #define BSEMAPHORE_DECL(name, taken) binary_semaphore_t name = _BSEMAPHORE_DATA(name, taken)

  Static semaphore initializer.

Typedefs

- typedef struct ch_binary_semaphore binary_semaphore_t

  Binary semaphore type.

Functions

- static void chBSemObjectInit (binary_semaphore_t *bsp, bool taken)

  Initializes a binary semaphore.
- static msg_t chBSemWait (binary_semaphore_t *bsp)

  Wait operation on the binary semaphore.
- static msg_t chBSemWaitS (binary_semaphore_t *bsp)

  Wait operation on the binary semaphore.
- static msg_t chBSemWaitTimeoutS (binary_semaphore_t *bsp, sysinterval_t timeout)

  Wait operation on the binary semaphore.
- static msg_t chBSemWaitTimeout (binary_semaphore_t *bsp, sysinterval_t timeout)

  Wait operation on the binary semaphore.
- static void chBSemResetI (binary_semaphore_t *bsp, bool taken)

  Reset operation on the binary semaphore.
- static void chBSemReset (binary_semaphore_t *bsp, bool taken)

  Reset operation on the binary semaphore.
- static void chBSemSignalI (binary_semaphore_t *bsp)

  Performs a signal operation on a binary semaphore.
- static void chBSemSignal (binary_semaphore_t *bsp)

  Performs a signal operation on a binary semaphore.
- static bool chBSemGetStateI (const binary_semaphore_t *bsp)

  Returns the binary semaphore current state.

8.3.1 Detailed Description

Binary semaphores structures and macros.

Binary semaphores related APIs and services.

Operation mode

Binary semaphores are implemented as a set of inline functions that use the existing counting semaphores primitives. The difference between counting and binary semaphores is that the counter of binary semaphores is not allowed to grow above the value 1. Repeated signal operation are ignored. A binary semaphore can thus have only two defined states:

- **Taken**, when its counter has a value of zero or lower than zero. A negative number represent the number of threads queued on the binary semaphore.

- **Not taken**, when its counter has a value of one.

Binary semaphores are different from mutexes because there is no concept of ownership, a binary semaphore can be taken by a thread and signaled by another thread or an interrupt handler, mutexes can only be taken and released by the same thread. Another difference is that binary semaphores, unlike mutexes, do not implement the priority inheritance protocol.
In order to use the binary semaphores APIs the `CH_CFG_USE_SEMAPHORES` option must be enabled in `chconf.h`.

8.4 chconf.h File Reference

Configuration file template.

Macros

Kernel parameters and options

- #define CH_CFG_NUM_THREADS 3
  Number of user threads in the application.

System timer settings

- #define CH_CFG_ST_RESOLUTION 32
  System time counter resolution.
- #define CH_CFG_ST_FREQUENCY 1000
  System tick frequency.
- #define CH_CFG_ST_TIMEDELTA 0
  Time delta constant for the tick-less mode.

Subsystem options

- #define CH_CFG_USE_SEMAPHORES TRUE
  Semaphores APIs.
- #define CH_CFG_USE_MUTEXES FALSE
  Mutexes APIs.
- #define CH_CFG_USE_EVENTS TRUE
  Events Flags APIs.
- #define CH_CFG_USE_MAILBOXES TRUE
  Mailboxes APIs.
- #define CH_CFG_USE_MEMCORE TRUE
  Core Memory Manager APIs.
- #define CH_CFG_USE_HEAP TRUE
  Heap Allocator APIs.
- #define CH_CFG_USE_MEMPOOLS TRUE
  Memory Pools Allocator APIs.
- #define CH_CFG_USE_OBJ_FIFOS TRUE
  Objects FIFOs APIs.
- #define CH_CFG_USE_PIPES TRUE
  Pipes APIs.
- #define CH_CFG_MEMCORE_SIZE 0
  Managed RAM size.

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
  System assertions.
- #define CH_DBG_ENABLE_STACK_CHECK TRUE
  Stack check.

Kernel hooks

- #define CH_CFG_SYSTEM_INIT_HOOK()
  System initialization hook.
- #define CH_CFG_THREAD_EXT_FIELDS /∗ Add threads custom fields here.∗/
  Threads descriptor structure extension.
- #define CH_CFG_THREAD_EXT_INIT_HOOK(tr)
  Threads initialization hook.
- #define CH_CFG_IDLE_ENTER_HOOK()
  Idle thread enter hook.
- #define CH_CFG_IDLE_LEAVE_HOOK()
  Idle thread leave hook.
- #define CH_CFG_SYSTEM_HALT_HOOK(reason)
  System halt hook.

8.4.1 Detailed Description

Configuration file template.

A copy of this file must be placed in each project directory, it contains the application specific kernel settings.

8.5 chfactory.c File Reference

ChibiOS objects factory and registry code.

```
#include <string.h>
#include "ch.h"
```

Functions

- void _factory_init (void)

  Initializes the objects factory.
- registered_object_t * chFactoryRegisterObject (const char *name, void *objp)

  Registers a generic object.
- registered_object_t * chFactoryFindObject (const char *name)

  Retrieves a registered object.
- registered_object_t * chFactoryFindObjectByPointer (void *objp)

  Retrieves a registered object by pointer.
- void chFactoryReleaseObject (registered_object_t *rop)

  Releases a registered object.
- dyn_buffer_t * chFactoryCreateBuffer (const char *name, size_t size)

  Creates a generic dynamic buffer object.
- dyn_buffer_t * chFactoryFindBuffer (const char *name)

  Retrieves a dynamic buffer object.
- void chFactoryReleaseBuffer (dyn_buffer_t *dbp)

  Releases a dynamic buffer object.
- dyn_semaphore_t * chFactoryCreateSemaphore (const char *name, cnt_t n)

  Creates a dynamic semaphore object.
- dyn_semaphore_t * chFactoryFindSemaphore (const char *name)

  Retrieves a dynamic semaphore object.
- void chFactoryReleaseSemaphore (dyn_semaphore_t *dsp)

  Releases a dynamic semaphore object.
- dyn_mailbox_t * chFactoryCreateMailbox (const char *name, size_t n)

  Creates a dynamic mailbox object.
- dyn_mailbox_t * chFactoryFindMailbox (const char *name)

  Retrieves a dynamic mailbox object.
- void chFactoryReleaseMailbox (dyn_mailbox_t *dmp)

  Releases a dynamic mailbox object.
- dyn_objects_fifo_t * chFactoryCreateObjectsFIFO (const char *name, size_t objsize, size_t objn, unsigned objalign)

  Creates a dynamic "objects FIFO" object.
- dyn_objects_fifo_t * chFactoryFindObjectsFIFO (const char *name)

  Retrieves a dynamic "objects FIFO" object.
- void chFactoryReleaseObjectsFIFO (dyn_objects_fifo_t *dofp)

  Releases a dynamic "objects FIFO" object.
- dyn_pipe_t * chFactoryCreatePipe (const char *name, size_t size)

  Creates a dynamic pipe object.
- dyn_pipe_t * chFactoryFindPipe (const char *name)

  Retrieves a dynamic pipe object.
- void chFactoryReleasePipe (dyn_pipe_t *dpp)

  Releases a dynamic pipe object.

Variables

- objects_factory_t ch_factory

  Factory object static instance.

8.5.1 Detailed Description

ChibiOS objects factory and registry code.

8.6 chfactory.h File Reference

ChibiOS objects factory structures and macros.

Data Structures

- struct ch_dyn_element
  Type of a dynamic object list element.
- struct ch_dyn_list
  Type of a dynamic object list.
- struct ch_registered_static_object
  Type of a registered object.
- struct ch_dyn_object
  Type of a dynamic buffer object.
- struct ch_dyn_semaphore
  Type of a dynamic semaphore.
- struct ch_dyn_mailbox
  Type of a dynamic buffer object.
- struct ch_dyn_objects_fifo
  Type of a dynamic buffer object.
- struct ch_dyn_pipe
  Type of a dynamic pipe object.
- struct ch_objects_factory
  Type of the factory main object.

Macros

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

- #define CH_CFG_FACTORY_MAILBOXES FALSE

  Enables factory for mailboxes.
- #define CH_CFG_FACTORY_OBJ_FIFOS FALSE

  Enables factory for objects FIFOs.
- #define CH_CFG_FACTORY_PIPES FALSE

  Enables factory for Pipes.

Typedefs

- typedef struct ch_dyn_element dyn_element_t

  Type of a dynamic object list element.
- typedef struct ch_dyn_list dyn_list_t

  Type of a dynamic object list.
- typedef struct ch_registered_static_object registered_object_t

  Type of a registered object.
- typedef struct ch_dyn_object dyn_buffer_t

  Type of a dynamic buffer object.
- typedef struct ch_dyn_semaphore dyn_semaphore_t

  Type of a dynamic semaphore.
- typedef struct ch_dyn_mailbox dyn_mailbox_t

  Type of a dynamic buffer object.
- typedef struct ch_dyn_objects_fifo dyn_objects_fifo_t

  Type of a dynamic buffer object.
- typedef struct ch_dyn_pipe dyn_pipe_t

  Type of a dynamic pipe object.
- typedef struct ch_objects_factory objects_factory_t

  Type of the factory main object.

Functions

- void _factory_init (void)

  Initializes the objects factory.
- registered_object_t * chFactoryRegisterObject (const char *name, void *objp)

  Registers a generic object.
- registered_object_t * chFactoryFindObject (const char *name)

  Retrieves a registered object.
- registered_object_t * chFactoryFindObjectByPointer (void *objp)

  Retrieves a registered object by pointer.
- void chFactoryReleaseObject (registered_object_t *rop)

  Releases a registered object.
- dyn_buffer_t * chFactoryCreateBuffer (const char *name, size_t size)

  Creates a generic dynamic buffer object.
- dyn_buffer_t * chFactoryFindBuffer (const char *name)

  Retrieves a dynamic buffer object.
- void chFactoryReleaseBuffer (dyn_buffer_t *dbp)

  Releases a dynamic buffer object.
- dyn_semaphore_t * chFactoryCreateSemaphore (const char *name, 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.

8.6.1 Detailed Description

ChibiOS objects factory structures and macros.

8.7 chlib.h File Reference

ChibiOS/LIB main include file.

```
#include "chbsem.h"
#include "chmboxes.h"
#include "chmemcore.h"
#include "chmemheaps.h"
#include "chmempools.h"
#include "chobjfifos.h"
#include "chpipes.h"
#include "chfactory.h"
```

Macros

- #define _CHIBIOS_OSLIB_

  ChibiOS/LIB identification macro.
- #define CH_OSLIB_STABLE 1

  Stable release flag.

ChibiOS/LIB version identification

- #define CH_OSLIB_VERSION "1.1.2"

  OS Library version string.
- #define CH_OSLIB_MAJOR 1

  OS Library version major number.
- #define CH_OSLIB_MINOR 1

  OS Library version minor number.
- #define CH_OSLIB_PATCH 2

  OS Library version patch number.

8.7.1 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.

8.8 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.

8.8.1 Detailed Description

Mailboxes code.

8.9 chmboxes.h File Reference

Mailboxes macros and structures.

Data Structures

- struct mailbox_t

  Structure representing a mailbox object.

Macros

- #define _MAILBOX_DATA(name, buffer, size)

  Data part of a static mailbox initializer.
- #define MAILBOX_DECL(name, buffer, size) mailbox_t name = _MAILBOX_DATA(name, buffer, size)

  Static mailbox initializer.

Functions

- void chMBObjectInit (mailbox_t *mbp, msg_t *buf, size_t n)

  Initializes a `mailbox_t` object.
- void chMBReset (mailbox_t *mbp)

  Resets a `mailbox_t` object.
- void chMBResetI (mailbox_t *mbp)

  Resets a `mailbox_t` object.
- msg_t chMBPostTimeout (mailbox_t *mbp, msg_t msg, sysinterval_t timeout)

  Posts a message into a mailbox.
- msg_t chMBPostTimeoutS (mailbox_t *mbp, msg_t msg, sysinterval_t timeout)

  Posts a message into a mailbox.
- msg_t chMBPostI (mailbox_t *mbp, msg_t msg)

Posts a message into a mailbox.

- msg_t chMBPostAheadTimeout (mailbox_t *mbp, msg_t msg, sysinterval_t timeout)

  Posts an high priority message into a mailbox.
- msg_t chMBPostAheadTimeoutS (mailbox_t *mbp, msg_t msg, sysinterval_t timeout)

  Posts an high priority message into a mailbox.
- msg_t chMBPostAheadI (mailbox_t *mbp, msg_t msg)

  Posts an high priority message into a mailbox.
- msg_t chMBFetchTimeout (mailbox_t *mbp, msg_t *msgp, sysinterval_t timeout)

  Retrieves a message from a mailbox.
- msg_t chMBFetchTimeoutS (mailbox_t *mbp, msg_t *msgp, sysinterval_t timeout)

  Retrieves a message from a mailbox.
- msg_t chMBFetchI (mailbox_t *mbp, msg_t *msgp)

  Retrieves a message from a mailbox.
- static size_t 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.

8.9.1 Detailed Description

Mailboxes macros and structures.

8.10 chmemcore.c File Reference

Core memory manager code.

```
#include "ch.h"
```

Functions

- void _core_init (void)

  Low level memory manager initialization.
- void * chCoreAllocAlignedWithOffsetI (size_t size, unsigned align, size_t offset)

  Allocates a memory block.
- void * chCoreAllocAlignedWithOffset (size_t size, unsigned align, size_t offset)

  Allocates a memory block.
- size_t chCoreGetStatusX (void)

  Core memory status.

Variables

- memcore_t ch_memcore

  Memory core descriptor.

8.10.1 Detailed Description

Core memory manager code.

8.11 chmemcore.h File Reference

Core memory manager macros and structures.

Data Structures

- struct memcore_t

  Type of memory core object.

Macros

- #define CH_CFG_MEMCORE_SIZE 0

  Managed RAM size.

Typedefs

- typedef void *(* memgetfunc_t) (size_t size, unsigned align)

  Memory get function.
- typedef void *(* memgetfunc2_t) (size_t size, unsigned align, size_t offset)

  Enhanced memory get function.

Functions

- void _core_init (void)

  Low level memory manager initialization.
- void * chCoreAllocAlignedWithOffsetI (size_t size, unsigned align, size_t offset)

  Allocates a memory block.
- void * chCoreAllocAlignedWithOffset (size_t size, unsigned align, size_t offset)

  Allocates a memory block.
- size_t chCoreGetStatusX (void)

  Core memory status.
- static void * chCoreAllocAlignedI (size_t size, unsigned align)

  Allocates a memory block.
- static void * chCoreAllocAligned (size_t size, unsigned align)

  Allocates a memory block.
- static void * chCoreAllocI (size_t size)

  Allocates a memory block.
- static void * chCoreAlloc (size_t size)

  Allocates a memory block.

8.11.1 Detailed Description

Core memory manager macros and structures.

8.12 chmemheaps.c File Reference

Memory heaps code.

```
#include "ch.h"
```

Functions

- void _heap_init (void)

  Initializes the default heap.
- void chHeapObjectInit (memory_heap_t *heapp, void *buf, size_t size)

  Initializes a memory heap from a static memory area.
- void * chHeapAllocAligned (memory_heap_t *heapp, size_t size, unsigned align)

  Allocates a block of memory from the heap by using the first-fit algorithm.
- void chHeapFree (void *p)

  Frees a previously allocated memory block.
- size_t chHeapStatus (memory_heap_t *heapp, size_t *totalp, size_t *largestp)

  Reports the heap status.

Variables

- static memory_heap_t default_heap

  Default heap descriptor.

8.12.1 Detailed Description

Memory heaps code.

8.13 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.

8.13.1 Detailed Description

Memory heaps macros and structures.

8.14 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.

8.14.1 Detailed Description

Memory Pools code.

8.15 chmempools.h File Reference

Memory Pools macros and structures.

Data Structures

- struct pool_header

  Memory pool free object header.
- struct memory_pool_t

  Memory pool descriptor.
- struct guarded_memory_pool_t

  Guarded memory pool descriptor.

Macros

- #define _MEMORYPOOL_DATA(name, size, align, provider) {NULL, size, align, provider}

  Data part of a static memory pool initializer.
- #define MEMORYPOOL_DECL(name, size, align, provider) memory_pool_t name = _MEMORYPOOL_D↩ ATA(name, size, align, provider)

  Static memory pool initializer.
- #define _GUARDEDMEMORYPOOL_DATA(name, size, align)

  Data part of a static guarded memory pool initializer.
- #define GUARDEDMEMORYPOOL_DECL(name, size, align) guarded_memory_pool_t name = _GUARD↩ EDMEMORYPOOL_DATA(name, size, align)

  Static guarded memory pool initializer.

Functions

- void chPoolObjectInitAligned (memory_pool_t *mp, size_t size, unsigned align, memgetfunc_t provider)

  Initializes an empty memory pool.
- void chPoolLoadArray (memory_pool_t *mp, void *p, size_t n)

  Loads a memory pool with an array of static objects.
- void * 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.

8.15.1 Detailed Description

Memory Pools macros and structures.

8.16 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.

8.16.1 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.

8.17 chpipes.c File Reference

Pipes code.

```
#include <string.h>
#include "ch.h"
```

Functions

- static size_t pipe_write (pipe_t *pp, const uint8_t *bp, size_t n)
  Non-blocking pipe write.
- static size_t pipe_read (pipe_t *pp, uint8_t *bp, size_t n)
  Non-blocking pipe read.
- void chPipeObjectInit (pipe_t *pp, uint8_t *buf, size_t n)
  Initializes a `mailbox_t` object.
- void chPipeReset (pipe_t *pp)
  Resets a `pipe_t` object.
- size_t chPipeWriteTimeout (pipe_t *pp, const uint8_t *bp, size_t n, sysinterval_t timeout)
  Pipe write with timeout.
- size_t chPipeReadTimeout (pipe_t *pp, uint8_t *bp, size_t n, sysinterval_t timeout)
  Pipe read with timeout.

8.17.1 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.

8.18 chpipes.h File Reference

Pipes macros and structures.

Data Structures

- struct pipe_t

  Structure representing a pipe object.

Macros

- #define _PIPE_DATA(name, buffer, size)

  Data part of a static pipe initializer.
- #define PIPE_DECL(name, buffer, size) pipe_t name = _PIPE_DATA(name, buffer, size)

  Static pipe initializer.

Functions

- void chPipeObjectInit (pipe_t *pp, uint8_t *buf, size_t n)

  Initializes a `mailbox_t` object.
- void chPipeReset (pipe_t *pp)

  Resets a `pipe_t` object.
- size_t chPipeWriteTimeout (pipe_t *pp, const uint8_t *bp, size_t n, sysinterval_t timeout)

  Pipe write with timeout.
- size_t chPipeReadTimeout (pipe_t *pp, uint8_t *bp, size_t n, sysinterval_t timeout)

  Pipe read with timeout.
- static size_t chPipeGetSize (const pipe_t *pp)

  Returns the pipe buffer size as number of bytes.
- static size_t chPipeGetUsedCount (const pipe_t *pp)

  Returns the number of used byte slots into a pipe.
- static size_t chPipeGetFreeCount (const pipe_t *pp)

  Returns the number of free byte slots into a pipe.
- static void chPipeResume (pipe_t *pp)

  Terminates the reset state.

8.18.1 Detailed Description

Pipes macros and structures.

Index