ChibiOS/NIL

1.1.3

Reference Manual

Sun Jan 1 2017 18:09:05

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

Data Structure Index

3.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 4

File Index

4.1 File List

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

Chapter 5

Module Documentation

5.1 NIL Kernel

5.1.1 Detailed Description

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

Modules

- Configuration
- Kernel Types
- API
- Port Layer
- Timer Interface

5.2 Configuration

5.2.1 Detailed Description

Kernel related settings and hooks.

Kernel parameters and options

- #define NIL_CFG_NUM_THREADS 1

  Number of user threads in the application.

System timer settings

- #define NIL_CFG_ST_RESOLUTION 32

  System time counter resolution.
- #define NIL_CFG_ST_FREQUENCY 50000

  System tick frequency.
- #define NIL_CFG_ST_TIMEDELTA 2

  Time delta constant for the tick-less mode.

Subsystem options

- #define NIL_CFG_USE_EVENTS TRUE

  Events Flags APIs.

Debug options

- #define NIL_CFG_ENABLE_ASSERTS FALSE

  System assertions.
- #define NIL_CFG_ENABLE_STACK_CHECK FALSE

  Stack check.

Kernel hooks

- #define NIL_CFG_SYSTEM_INIT_HOOK()

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

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

  Threads initialization hook.
- #define NIL_CFG_IDLE_ENTER_HOOK()

  Idle thread enter hook.
- #define NIL_CFG_IDLE_LEAVE_HOOK()

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

  System halt hook.

5.2.2 Macro Definition Documentation

5.2.2.1 #define NIL_CFG_NUM_THREADS 1

Number of user threads in the application.

Note

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

5.2.2.2 #define NIL_CFG_ST_RESOLUTION 32

System time counter resolution.

Note

Allowed values are 16 or 32 bits.

5.2.2.3 #define NIL_CFG_ST_FREQUENCY 50000

System tick frequency.

Note

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

5.2.2.4 #define NIL_CFG_ST_TIMEDELTA 2

Time delta constant for the tick-less mode.

Note

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

5.2.2.5 #define NIL_CFG_USE_EVENTS TRUE

Events Flags APIs.

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

Note

The default is `TRUE`.

5.2.2.6 #define NIL_CFG_ENABLE_ASSERTS FALSE

System assertions.

5.2.2.7 #define NIL_CFG_ENABLE_STACK_CHECK FALSE

Stack check.

5.2.2.8 #define NIL_CFG_SYSTEM_INIT_HOOK( )

Value:

```
{                                          \
}
```

System initialization hook.

5.2.2.9 #define NIL_CFG_THREAD_EXT_FIELDS /* Add threads custom fields here.*/

Threads descriptor structure extension.

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

5.2.2.10 #define NIL_CFG_THREAD_EXT_INIT_HOOK(*tr*)

Value:

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

Threads initialization hook.

5.2.2.11 #define NIL_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.

5.2.2.12 #define NIL_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.

5.2.2.13 #define NIL_CFG_SYSTEM_HALT_HOOK(*reason*)

Value:

```
{                                   \
}
```

System halt hook.

5.3 Kernel Types

5.3.1 Detailed Description

Macros

- #define ROMCONST const

    ROM constant modifier.
- #define NOINLINE __attribute__((noinline))

    Makes functions not inlineable.
- #define PORT_THD_FUNCTION(tname, arg) __attribute__((noreturn)) void tname(void ∗arg)

    Optimized thread function declaration macro.
- #define PACKED_VAR __attribute__((packed))

    Packed variable specifier.

Common constants

- #define FALSE 0

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

    Generic 'true' boolean constant.

Typedefs

- typedef uint32_t syssts_t
- typedef uint32_t rtcnt_t
- typedef uint8_t tstate_t
- typedef int32_t msg_t
- typedef uint32_t eventmask_t
- typedef int32_t cnt_t
- typedef uint32_t ucnt_t
- typedef uint32_t systime_t

    Type of system time.

5.3.2 Macro Definition Documentation

5.3.2.1 #define FALSE 0

Generic 'false' boolean constant.

5.3.2.2 #define TRUE 1

Generic 'true' boolean constant.

5.3.2.3 #define ROMCONST const

ROM constant modifier.

Note

It is set to use the "const" keyword in this port.

5.3.2.4 #define NOINLINE __attribute__((noinline))

Makes functions not inlineable.

Note

If the compiler does not support such attribute then the realtime counter precision could be degraded.

5.3.2.5 #define PORT_THD_FUNCTION(*tname*, *arg*) __attribute__((noreturn)) void tname(void ∗arg)

Optimized thread function declaration macro.

5.3.2.6 #define PACKED_VAR __attribute__((packed))

Packed variable specifier.

5.3.3 Typedef Documentation

5.3.3.1 typedef uint32_t syssts_t

System status word.

5.3.3.2 typedef uint32_t rtcnt_t

Realtime counter.

5.3.3.3 typedef uint8_t tstate_t

Thread state.

5.3.3.4 typedef int32_t msg_t

Inter-thread message.

5.3.3.5 typedef uint32_t eventmask_t

Mask of event identifiers.

5.3.3.6 typedef int32_t cnt_t

Generic signed counter.

5.3.3.7 typedef uint32_t ucnt_t

Generic unsigned counter.

5.3.3.8 typedef uint32_t systime_t

Type of system time.

5.4 API

5.4.1 Detailed Description

Macros

- #define _CHIBIOS_NIL_

  ChibiOS/NIL identification macro.
- #define CH_KERNEL_STABLE 1

  Stable release flag.
- #define NIL_CFG_NUM_THREADS 2

  Number of user threads in the application.
- #define NIL_CFG_ST_RESOLUTION 32

  System time counter resolution.
- #define NIL_CFG_ST_FREQUENCY 100

  System tick frequency.
- #define NIL_CFG_ST_TIMEDELTA 0

  Time delta constant for the tick-less mode.
- #define NIL_CFG_USE_EVENTS TRUE

  Events Flags APIs.
- #define NIL_CFG_ENABLE_ASSERTS FALSE

  System assertions.
- #define NIL_CFG_ENABLE_STACK_CHECK FALSE

  Stack check.
- #define NIL_CFG_SYSTEM_INIT_HOOK() {}

  System initialization hook.
- #define NIL_CFG_THREAD_EXT_FIELDS

  Threads descriptor structure extension.
- #define NIL_CFG_THREAD_EXT_INIT_HOOK(tr) {}

  Threads initialization hook.
- #define NIL_CFG_IDLE_ENTER_HOOK() {}

  Idle thread enter hook.
- #define NIL_CFG_IDLE_LEAVE_HOOK() {}

  Idle thread leave hook.
- #define NIL_CFG_SYSTEM_HALT_HOOK(reason) {}

  System halt hook.

ChibiOS/NIL version identification

- #define CH_KERNEL_VERSION "1.1.3"

  Kernel version string.
- #define CH_KERNEL_MAJOR 1

  Kernel version major number.
- #define CH_KERNEL_MINOR 1

  Kernel version minor number.
- #define CH_KERNEL_PATCH 3

  Kernel version patch number.

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 ((systime_t)-1)

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

    Infinite time specification for all functions with a timeout specification.

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_WTSEM (tstate_t)3

    On semaphore.
- #define NIL_STATE_WTOREVT (tstate_t)4

    Waiting for events.
- #define **NIL_THD_IS_READY**(tr) ((tr)->state == NIL_STATE_READY)
- #define **NIL_THD_IS_SLEEPING**(tr) ((tr)->state == NIL_STATE_SLEEPING)
- #define **NIL_THD_IS_SUSP**(tr) ((tr)->state == NIL_STATE_SUSP)
- #define **NIL_THD_IS_WTSEM**(tr) ((tr)->state == NIL_STATE_WTSEM)
- #define **NIL_THD_IS_WTOREVT**(tr) ((tr)->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[NIL_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.

Working Areas and Alignment

- #define THD_ALIGN_STACK_SIZE(n) ((((n) - 1U) | (sizeof(stkalign_t) - 1U)) + 1U)

  Enforces a correct alignment for a stack area size value.
- #define THD_WORKING_AREA_SIZE(n) THD_ALIGN_STACK_SIZE(PORT_WA_SIZE(n))

  Calculates the total Working Area size.
- #define THD_WORKING_AREA(s, n) stkalign_t s[THD_WORKING_AREA_SIZE(n) / sizeof(stkalign_t)]

  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() PORT_IRQ_PROLOGUE()

  IRQ handler enter code.
- #define CH_IRQ_EPILOGUE() PORT_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 S2ST(sec) ((systime_t)((uint32_t)(sec) ∗ (uint32_t)NIL_CFG_ST_FREQUENCY))

  Seconds to system ticks.
- #define MS2ST(msec)

  Milliseconds to system ticks.
- #define US2ST(usec)

  Microseconds to system ticks.

Time conversion utilities for the realtime counter

- #define S2RTC(freq, sec) ((freq) ∗ (sec))

  Seconds to realtime counter.
- #define MS2RTC(freq, msec) (rtcnt_t)((((freq) + 999UL) / 1000UL) ∗ (msec))

  Milliseconds to realtime counter.
- #define US2RTC(freq, usec) (rtcnt_t)((((freq) + 999999UL) / 1000000UL) ∗ (usec))

  Microseconds to realtime counter.

Macro Functions

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

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

  Enters the kernel lock mode.
- #define chSysEnable() port_enable()

  Enters the kernel lock mode.
- #define chSysLock() port_lock()

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

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

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

  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(sec) chThdSleep(S2ST(sec))

  Delays the invoking thread for the specified number of seconds.
- #define chThdSleepMilliseconds(msec) chThdSleep(MS2ST(msec))

  Delays the invoking thread for the specified number of milliseconds.
- #define chThdSleepMicroseconds(usec) chThdSleep(US2ST(usec))

  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 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 chSemGetCounterI(sp) ((sp)->cnt)

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

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

  Returns the elapsed time since the specified start time.
- #define chVTIsTimeWithinX(time, start, end) ((bool)((systime_t)((time) - (start)) < (systime_t)((end) - (start))))

  Checks if the specified time is within the specified time window.
- #define chDbgAssert(c, r)

  Condition assertion.

Typedefs

- typedef struct nil_thread thread_t

  Type of a structure representing a thread.
- typedef struct port_intctx intctx_t

  Type of internal context structure.
- typedef struct nil_semaphore 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_semaphore

  Structure representing a counting semaphore.
- struct nil_thread_cfg

  Structure representing a thread static configuration.
- struct nil_thread

  Structure representing a thread.
- struct nil_system

  System data structure.

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.
- 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.

- void chSchRescheduleS (void)

    Reschedules if needed.

- msg_t chSchGoSleepTimeoutS (tstate_t newstate, systime_t timeout)

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

- msg_t chThdSuspendTimeoutS (thread_reference_t ∗trp, systime_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 chThdSleep (systime_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 chSemWaitTimeout (semaphore_t ∗sp, systime_t timeout)

    Performs a wait operation on a semaphore with timeout specification.

- msg_t chSemWaitTimeoutS (semaphore_t ∗sp, systime_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, systime_t timeout)

    Waits for any of the specified events.

- eventmask_t chEvtWaitAnyTimeoutS (eventmask_t mask, systime_t timeout)

    Waits for any of the specified events.

Variables

- nil_system_t nil

    System data structures.

- stkalign_t __main_thread_stack_base__

5.4.2 Macro Definition Documentation

5.4.2.1 #define _CHIBIOS_NIL_

ChibiOS/NIL identification macro.

5.4.2.2 #define CH_KERNEL_STABLE 1

Stable release flag.

5.4.2.3 #define CH_KERNEL_VERSION "1.1.3"

Kernel version string.

5.4.2.4 #define CH_KERNEL_MAJOR 1

Kernel version major number.

5.4.2.5 #define CH_KERNEL_MINOR 1

Kernel version minor number.

5.4.2.6 #define CH_KERNEL_PATCH 3

Kernel version patch number.

5.4.2.7 #define MSG_OK (msg_t)0

OK wakeup message.

5.4.2.8 #define MSG_TIMEOUT (msg_t)-1

Wake-up caused by a timeout condition.

5.4.2.9 #define MSG_RESET (msg_t)-2

Wake-up caused by a reset condition.

5.4.2.10 #define TIME_IMMEDIATE ((systime_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.

5.4.2.11 #define TIME_INFINITE ((systime_t)0)

Infinite time specification for all functions with a timeout specification.

5.4.2.12 #define NIL_STATE_READY (tstate_t)0

Thread ready or executing.

5.4.2.13 #define NIL_STATE_SLEEPING (tstate_t)1

Thread sleeping.

5.4.2.14 #define NIL_STATE_SUSP (tstate_t)2

Thread suspended.

5.4.2.15 #define NIL_STATE_WTSEM (tstate_t)3

On semaphore.

5.4.2.16 #define NIL_STATE_WTOREVT (tstate_t)4

Waiting for events.

5.4.2.17 #define ALL_EVENTS ((eventmask_t)-1)

All events allowed mask.

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

Returns an event mask from an event identifier.

5.4.2.19 #define NIL_CFG_NUM_THREADS 2

Number of user threads in the application.

Note

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

5.4.2.20 #define NIL_CFG_ST_RESOLUTION 32

System time counter resolution.

Note

Allowed values are 16 or 32 bits.

5.4.2.21 #define NIL_CFG_ST_FREQUENCY 100

System tick frequency.

Note

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

5.4.2.22 #define NIL_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.

5.4.2.23 #define NIL_CFG_USE_EVENTS TRUE

Events Flags APIs.

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

Note

The default is `TRUE`.

5.4.2.24 #define NIL_CFG_ENABLE_ASSERTS FALSE

System assertions.

5.4.2.25 #define NIL_CFG_ENABLE_STACK_CHECK FALSE

Stack check.

5.4.2.26 #define NIL_CFG_SYSTEM_INIT_HOOK( ) {}

System initialization hook.

5.4.2.27 #define NIL_CFG_THREAD_EXT_FIELDS

Threads descriptor structure extension.

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

5.4.2.28 #define NIL_CFG_THREAD_EXT_INIT_HOOK(*tr*) {}

Threads initialization hook.

5.4.2.29 #define NIL_CFG_IDLE_ENTER_HOOK( ) {}

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.

5.4.2.30 #define NIL_CFG_IDLE_LEAVE_HOOK( ) {}

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.

5.4.2.31 #define NIL_CFG_SYSTEM_HALT_HOOK(*reason*) {}

System halt hook.

5.4.2.32 #define THD_TABLE_BEGIN const thread_config_t nil_thd_configs[NIL_CFG_NUM_THREADS + 1] = {

Start of user threads table.

5.4.2.33 #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.

5.4.2.34 #define THD_TABLE_END

Value:

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

End of user threads table.

5.4.2.35 #define THD_ALIGN_STACK_SIZE(*n*) ((((n) - 1U) | (sizeof(stkalign_t) - 1U)) + 1U)

Enforces a correct alignment for a stack area size value.

Parameters

in	*n*	the stack size to be aligned to the next stack alignment boundary

Returns

The aligned stack size.

Function Class:

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

5.4.2.36 #define THD_WORKING_AREA_SIZE(*n*) THD_ALIGN_STACK_SIZE(PORT_WA_SIZE(n))

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.

5.4.2.37 #define THD_WORKING_AREA(*s,* *n*) stkalign_t s[THD_WORKING_AREA_SIZE(n) / sizeof(stkalign_t)]

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.

5.4.2.38 #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.

5.4.2.39 #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.

5.4.2.40 #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.

5.4.2.41 #define CH_IRQ_PROLOGUE( ) PORT_IRQ_PROLOGUE()

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.

5.4.2.42 #define CH_IRQ_EPILOGUE( ) 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.

5.4.2.43 #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.

5.4.2.44 #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.

5.4.2.45 #define S2ST(*sec*) ((systime_t)((uint32_t)(sec) ∗ (uint32_t)NIL_CFG_ST_FREQUENCY))

Seconds to system ticks.

Converts from seconds to system ticks number.

Note

The result is rounded upward to the next tick boundary.

Parameters

in	*sec*	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.

5.4.2.46 #define MS2ST(*msec*)

Value:

```
((systime_t)((((uint32_t)(msec)) *                              \
                ((uint32_t)NIL_CFG_ST_FREQUENCY)) + 999UL) / 1000UL))
```

Milliseconds to system ticks.

Converts from milliseconds to system ticks number.

Note

The result is rounded upward to the next tick boundary.

Parameters

in	*msec*	number of milliseconds

Returns

The number of ticks.

Function Class:

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

5.4.2.47 #define US2ST(*usec*)

Value:

```
((systime_t)((((uint32_t)(usec)) *                              \
                ((uint32_t)NIL_CFG_ST_FREQUENCY)) + 999999UL) / 1000000UL))
```

Microseconds to system ticks.

Converts from microseconds to system ticks number.

Note

The result is rounded upward to the next tick boundary.

Parameters

`in`	*usec*	number of microseconds

Returns

The number of ticks.

Function Class:

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

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

Seconds to realtime counter.

Converts from seconds to realtime counter cycles.

Note

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

Parameters

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

Returns

The number of cycles.

Function Class:

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

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

Milliseconds to realtime counter.

Converts from milliseconds to realtime counter cycles.

Note

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

Parameters

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

Returns

The number of cycles.

Function Class:

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

5.4.2.50 #define US2RTC(*freq,* *usec*) (rtcnt_t)((((freq) + 999999UL) / 1000000UL) $*$ (usec))

Microseconds to realtime counter.

Converts from microseconds to realtime counter cycles.

Note

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

Parameters

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

Returns

The number of cycles.

Function Class:

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

5.4.2.51 #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.

5.4.2.52 #define chSysDisable( ) port_disable()

Enters the kernel lock mode.

Function Class:

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

5.4.2.53 #define chSysEnable( ) port_enable()

Enters the kernel lock mode.

Function Class:

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

5.4.2.54 #define chSysLock( ) port_lock()

Enters the kernel lock state.

Function Class:

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

5.4.2.55 #define chSysUnlock( ) port_unlock()

Leaves the kernel lock state.

Function Class:

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

5.4.2.56 #define chSysLockFromISR( ) port_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.

5.4.2.57 #define chSysUnlockFromISR( ) 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.

5.4.2.58 #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.

5.4.2.59 #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.

5.4.2.60 #define chThdSleepSeconds(*sec*) chThdSleep(S2ST(sec))

Delays the invoking thread for the specified number of seconds.

Note

The specified time is rounded up to a value allowed by the real system clock.
The maximum specified value is implementation dependent.

Parameters

`in`	*sec*	time in seconds, must be different from zero

Function Class:

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

5.4.2.61 #define chThdSleepMilliseconds(*msec*) chThdSleep(MS2ST(msec))

Delays the invoking thread for the specified number of milliseconds.

Note

The specified time is rounded up to a value allowed by the real system clock.
The maximum specified value is implementation dependent.

Parameters

`in`	*msec*	time in milliseconds, must be different from zero

Function Class:

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

5.4.2.62 #define chThdSleepMicroseconds(*usec*) chThdSleep(US2ST(usec))

Delays the invoking thread for the specified number of microseconds.

Note

The specified time is rounded up to a value allowed by the real system clock.
The maximum specified value is implementation dependent.

Parameters

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

Function Class:

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

5.4.2.63 #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.

5.4.2.64 #define chThdSleepUntilS(*abstime*)

Value:

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

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.

5.4.2.65 #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.

5.4.2.66 #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.

5.4.2.67 #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.

5.4.2.68 #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.

5.4.2.69 #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.

5.4.2.70 #define chVTTimeElapsedSinceX(*start*) ((systime_t)(chVTGetSystemTimeX() - (start)))

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.

5.4.2.71 #define chVTIsTimeWithinX(*time, start, end*) ((bool)((systime_t)((time) - (start)) < (systime_t)((end) - (start))))

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

Note

When start==end then the function returns always true because the whole time range is specified.
This function can be called from any context.

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.

5.4.2.72 #define chDbgAssert(*c,* *r*)

Value:

```
do {                                                       \
  /*lint -save -e506 -e774 [2.1, 14.3] Can be a constant by design.*/       \
  if (NIL_CFG_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 `NIL_CFG_ENABLE_ASSERTS` switch is specified in `nilconf.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.

5.4.3 Typedef Documentation

5.4.3.1 typedef struct nil_thread thread_t

Type of a structure representing a thread.

Note

It is required as an early definition.

5.4.3.2 typedef struct port_intctx intctx_t

Type of internal context structure.

5.4.3.3 typedef struct nil_semaphore semaphore_t

Type of a structure representing a semaphore.

5.4.3.4 typedef void(∗ tfunc_t) (void ∗p)

Thread function.

5.4.3.5 typedef struct nil_thread_cfg thread_config_t

Type of a structure representing a thread static configuration.

5.4.3.6 typedef thread_t∗ thread_reference_t

Type of a thread reference.

5.4.3.7 typedef struct nil_system nil_system_t

Type of a structure representing the system.

5.4.4 Function Documentation

5.4.4.1 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.

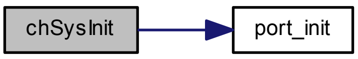
Note

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:



5.4.4.2 void chSysHalt (const char ∗ *reason*)

Halts the system.

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

Note

Can be invoked from any system state.

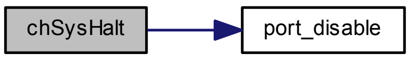
Parameters

`in`	*reason*	pointer to an error string

Function Class:

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

Here is the call graph for this function:



5.4.4.3 void chSysTimerHandlerI (void)

Time management handler.

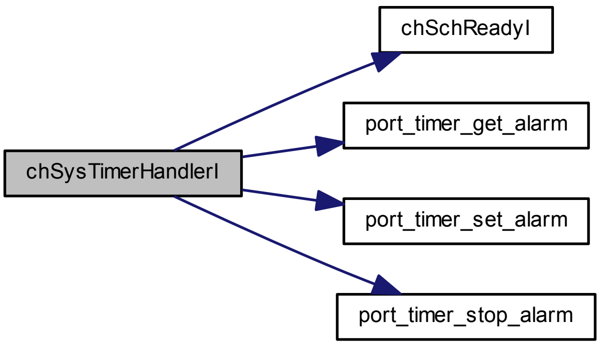
Note

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:



5.4.4.4 void chSysUnconditionalLock (void)

Unconditionally enters the kernel lock state.

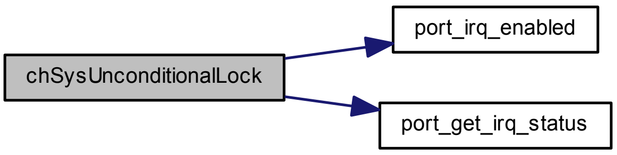
Note

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.

Here is the call graph for this function:



5.4.4.5 void chSysUnconditionalUnlock (void)

Unconditionally leaves the kernel lock state.

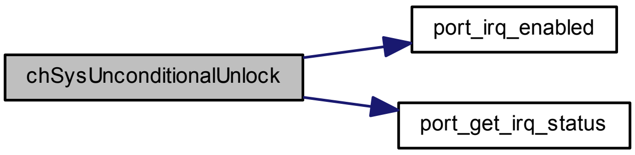
Note

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.

Here is the call graph for this function:



5.4.4.6 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.

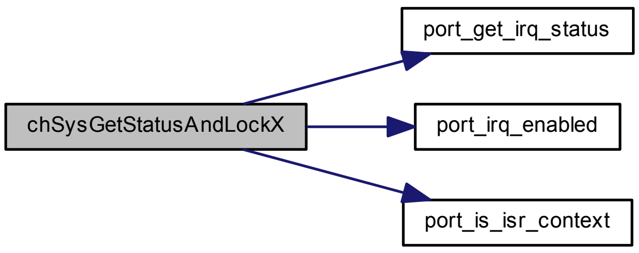
Returns

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.

Here is the call graph for this function:



5.4.4.7 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.

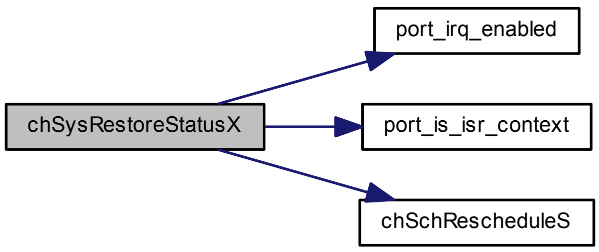
Parameters

`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:



5.4.4.8 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.

5.4.4.9 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`.

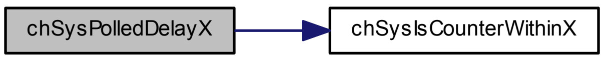
Parameters

`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:



5.4.4.10 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.

5.4.4.11 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.

5.4.4.12 msg_t chSchGoSleepTimeoutS (tstate_t *newstate,* systime_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.

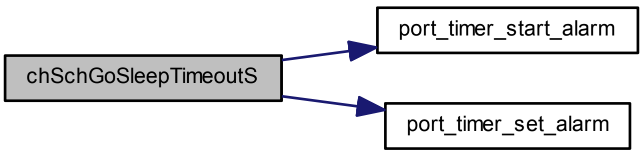
Return values

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:



5.4.4.13 msg_t chThdSuspendTimeoutS (thread_reference_t * *trp,* systime_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.

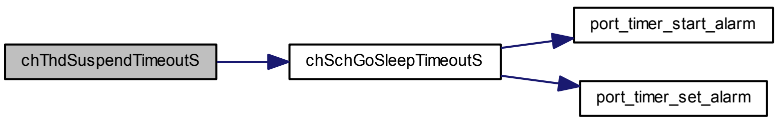
Returns

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:



5.4.4.14 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.

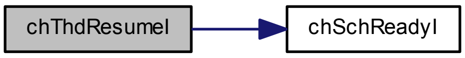
Parameters

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:



5.4.4.15 void chThdSleep (systime_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.

5.4.4.16 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.

5.4.4.17 msg_t chSemWaitTimeout (semaphore_t * *sp*, systime_t *timeout*)

Performs a wait operation on a semaphore with timeout specification.

Parameters

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

Returns

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

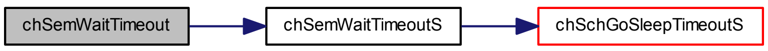
Return values

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:



5.4.4.18 msg_t chSemWaitTimeoutS (semaphore_t * *sp*, systime_t *timeout*)

Performs a wait operation on a semaphore with timeout specification.

Parameters

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

Returns

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

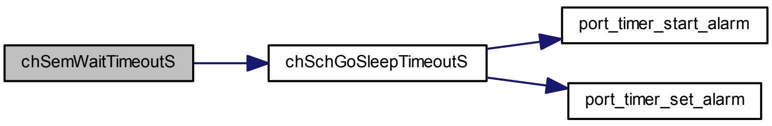
Return values

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:



5.4.4.19 void chSemSignal (**semaphore_t** ∗ *sp*)

Performs a signal operation on a semaphore.

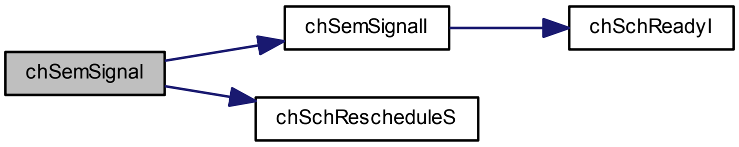
Parameters

`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:



5.4.4.20 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.

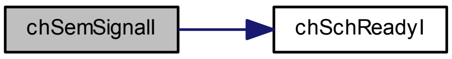
Parameters

`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:



5.4.4.21 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.

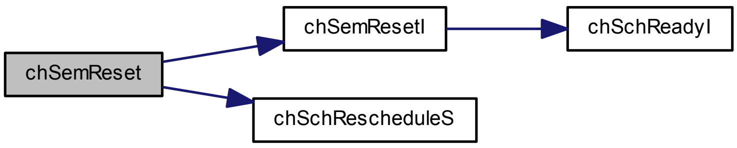
Parameters

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:



5.4.4.22 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.

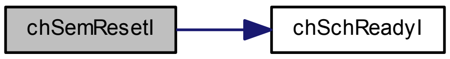
Parameters

`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:



5.4.4.23 void chEvtSignal (**thread_t** ∗ *tp,* **eventmask_t** *mask*)

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

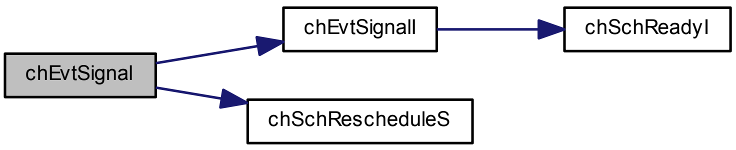
Parameters

`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:



5.4.4.24 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.

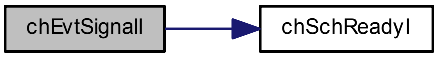
Parameters

`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:



5.4.4.25 eventmask_t chEvtWaitAnyTimeout (eventmask_t *mask,* systime_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_IMMEDIATE* immediate timeout. • *TIME_INFINITE* no timeout.

Returns

The mask of the served and cleared events.

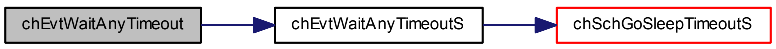
Return values

0	if the operation has timed out.

Function Class:

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

Here is the call graph for this function:



5.4.4.26 eventmask_t chEvtWaitAnyTimeoutS (eventmask_t *mask,* systime_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_IMMEDIATE* immediate timeout. • *TIME_INFINITE* no timeout.

Returns

The mask of the served and cleared events.

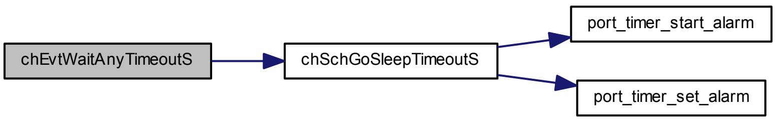
Return values

0	if the operation has timed out.

Function Class:

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:



5.4.5 Variable Documentation

5.4.5.1 nil_system_t nil

System data structures.

5.4.5.2 stkalign_t __main_thread_stack_base__

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

5.5 Port Layer

5.5.1 Detailed Description

Macros

- #define PORT_INT_REQUIRED_STACK 32

  Per-thread stack overhead for interrupts servicing.
- #define PORT_USE_ALT_TIMER FALSE

  Enables an alternative timer implementation.
- #define PORT_SETUP_CONTEXT(tp, wend, pf, arg)

  Platform dependent thread stack setup.
- #define PORT_WA_SIZE(n)

  Computes the thread working area global size.
- #define PORT_IRQ_IS_VALID_PRIORITY(n) false

  Priority level verification macro.
- #define PORT_IRQ_IS_VALID_KERNEL_PRIORITY(n) false

  Priority level verification macro.
- #define PORT_IRQ_PROLOGUE()

  IRQ prologue code.
- #define PORT_IRQ_EPILOGUE() _port_irq_epilogue()

  IRQ epilogue code.
- #define PORT_IRQ_HANDLER(id) void id(void)

  IRQ handler function declaration.
- #define PORT_FAST_IRQ_HANDLER(id) void id(void)

  Fast IRQ handler function declaration.
- #define port_switch(ntp, otp)

  Performs a context switch between two threads.

Architecture and Compiler

- #define PORT_ARCHITECTURE_XXX

  Macro defining the port architecture.
- #define PORT_ARCHITECTURE_NAME "XXX"

  Name of the implemented architecture.
- #define PORT_CORE_VARIANT_NAME "XXXX-Y"

  Name of the architecture variant.
- #define PORT_COMPILER_NAME "GCC " __VERSION__

  Compiler name and version.
- #define PORT_INFO "port description"

  Port-specific information string.
- #define PORT_SUPPORTS_RT FALSE

  This port supports a realtime counter.

Typedefs

- typedef uint64_t stkalign_t

  Type of stack and memory alignment enforcement.

Data Structures

- struct port_extctx

  Interrupt saved context.
- struct port_intctx

  System saved context.

Functions

- static void port_init (void)

  Port-related initialization code.
- static syssts_t port_get_irq_status (void)

  Returns a word encoding the current interrupts status.
- static bool port_irq_enabled (syssts_t sts)

  Checks the interrupt status.
- static bool port_is_isr_context (void)

  Determines the current execution context.
- static void port_lock (void)

  Kernel-lock action.
- static void port_unlock (void)

  Kernel-unlock action.
- static void port_lock_from_isr (void)

  Kernel-lock action from an interrupt handler.
- static void port_unlock_from_isr (void)

  Kernel-unlock action from an interrupt handler.
- static void port_disable (void)

  Disables all the interrupt sources.
- static void port_suspend (void)

  Disables the interrupt sources below kernel-level priority.
- static void port_enable (void)

  Enables all the interrupt sources.
- static void port_wait_for_interrupt (void)

  Enters an architecture-dependent IRQ-waiting mode.
- static rtcnt_t port_rt_get_counter_value (void)

  Returns the current value of the realtime counter.

5.5.2 Macro Definition Documentation

5.5.2.1 #define PORT_ARCHITECTURE_XXX

Macro defining the port architecture.

5.5.2.2 #define PORT_ARCHITECTURE_NAME "XXX"

Name of the implemented architecture.

5.5.2.3 #define PORT_CORE_VARIANT_NAME "XXXX-Y"

Name of the architecture variant.

5.5.2.4 #define PORT_COMPILER_NAME "GCC " __VERSION__

Compiler name and version.

5.5.2.5 #define PORT_INFO "port description"

Port-specific information string.

5.5.2.6 #define PORT_SUPPORTS_RT FALSE

This port supports a realtime counter.

5.5.2.7 #define PORT_INT_REQUIRED_STACK 32

Per-thread stack overhead for interrupts servicing.

This constant is used in the calculation of the correct working area size.

5.5.2.8 #define PORT_USE_ALT_TIMER FALSE

Enables an alternative timer implementation.

Usually the port uses a timer interface defined in the file `nilcore_timer.h`, if this option is enabled then the file `nilcore_timer_alt.h` is included instead.

5.5.2.9 #define PORT_SETUP_CONTEXT(*tp,* *wend,* *pf,* *arg*)

Value:

```
do {                                \
  (void)(tp);                                                               \
  (void)(wend);                                                             \
  (void)(pf);                                                               \
  (void)(arg);                                                              \
} while (false)
```

Platform dependent thread stack setup.

This code usually setup the context switching frame represented by an `port_intctx` structure.

5.5.2.10 #define PORT_WA_SIZE(*n*)

Value:

```
(sizeof(struct port_intctx) +                               \
                          sizeof(struct port_extctx) +                      \
                          (size_t)(n) +                                     \
                          (size_t)(PORT_INT_REQUIRED_STACK))
```

Computes the thread working area global size.

Note

There is no need to perform alignments in this macro.

5.5.2.11 #define PORT_IRQ_IS_VALID_PRIORITY(*n*) false

Priority level verification macro.

5.5.2.12 #define PORT_IRQ_IS_VALID_KERNEL_PRIORITY(*n*) false

Priority level verification macro.

5.5.2.13 #define PORT_IRQ_PROLOGUE( )

IRQ prologue code.

This macro must be inserted at the start of all IRQ handlers enabled to invoke system APIs.

5.5.2.14 #define PORT_IRQ_EPILOGUE( ) _port_irq_epilogue()

IRQ epilogue code.

This macro must be inserted at the end of all IRQ handlers enabled to invoke system APIs.

5.5.2.15 #define PORT_IRQ_HANDLER(*id*) void id(void)

IRQ handler function declaration.

Note

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

5.5.2.16 #define PORT_FAST_IRQ_HANDLER(*id*) void id(void)

Fast IRQ handler function declaration.

Note

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

5.5.2.17 #define port_switch(*ntp*, *otp*)

Value:

```
do {                                             \
  (void)ntp;                                                                \
  (void)otp;                                                                \
  /*_port_switch(ntp, otp)*/                                                \
} while (false)
```

Performs a context switch between two threads.

This is the most critical code in any port, this function is responsible for the context switch between 2 threads.

Note

The implementation of this code affects **directly** the context switch performance so optimize here as much as you can.

Parameters

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

5.5.3 Typedef Documentation

5.5.3.1 typedef uint64_t stkalign_t

Type of stack and memory alignment enforcement.

5.5.4 Function Documentation

5.5.4.1 static void port_init (void) [inline], [static]

Port-related initialization code.

5.5.4.2 static syssts_t port_get_irq_status (void) [inline], [static]

Returns a word encoding the current interrupts status.

Returns

The interrupts status.

5.5.4.3 static bool port_irq_enabled (syssts_t *sts*) [inline], [static]

Checks the interrupt status.

Parameters

in	*sts*	the interrupt status word

Returns

The interrupt status.

Return values

false	the word specified a disabled interrupts status.
true	the word specified an enabled interrupts status.

5.5.4.4 static bool port_is_isr_context (void) [inline], [static]

Determines the current execution context.

Returns

The execution context.

Return values

false	not running in ISR mode.
true	running in ISR mode.

5.5.4.5 static void port_lock (void) `[inline]`, `[static]`

Kernel-lock action.

5.5.4.6 static void port_unlock (void) `[inline]`, `[static]`

Kernel-unlock action.

5.5.4.7 static void port_lock_from_isr (void) `[inline]`, `[static]`

Kernel-lock action from an interrupt handler.

5.5.4.8 static void port_unlock_from_isr (void) `[inline]`, `[static]`

Kernel-unlock action from an interrupt handler.

5.5.4.9 static void port_disable (void) `[inline]`, `[static]`

Disables all the interrupt sources.

5.5.4.10 static void port_suspend (void) `[inline]`, `[static]`

Disables the interrupt sources below kernel-level priority.

5.5.4.11 static void port_enable (void) `[inline]`, `[static]`

Enables all the interrupt sources.

5.5.4.12 static void port_wait_for_interrupt (void) `[inline]`, `[static]`

Enters an architecture-dependent IRQ-waiting mode.

The function is meant to return when an interrupt becomes pending. The simplest implementation is an empty function or macro but this would not take advantage of architecture-specific power saving modes.

5.5.4.13 static rtcnt_t port_rt_get_counter_value (void) `[inline]`, `[static]`

Returns the current value of the realtime counter.

Returns

The realtime counter value.

5.6 Timer Interface

5.6.1 Detailed Description

Functions

- static void port_timer_start_alarm (systime_t abstime)

  Starts the alarm.
- static void port_timer_stop_alarm (void)

  Stops the alarm interrupt.
- static void port_timer_set_alarm (systime_t abstime)

  Sets the alarm time.
- static systime_t port_timer_get_time (void)

  Returns the system time.
- static systime_t port_timer_get_alarm (void)

  Returns the current alarm time.

5.6.2 Function Documentation

5.6.2.1 static void port_timer_start_alarm (systime_t *abstime*) `[inline]`, `[static]`

Starts the alarm.

Note

Makes sure that no spurious alarms are triggered after this call.

Parameters

`in`	*abstime*	the time to be set for the first alarm

Function Class:

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

5.6.2.2 static void port_timer_stop_alarm (void) `[inline]`, `[static]`

Stops the alarm interrupt.

Function Class:

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

5.6.2.3 static void port_timer_set_alarm (systime_t *abstime*) `[inline]`, `[static]`

Sets the alarm time.

Parameters

`in`	*abstime*	the time to be set for the next alarm

Function Class:

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

5.6.2.4 static systime_t port_timer_get_time (void) [inline], [static]

Returns the system time.

Returns

The system time.

Function Class:

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

5.6.2.5 static systime_t port_timer_get_alarm (void) [inline], [static]

Returns the current alarm time.

Returns

The currently set alarm time.

Function Class:

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

Chapter 6

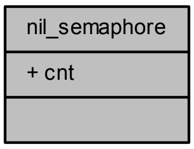
Data Structure Documentation

6.1 nil_semaphore Struct Reference

Structure representing a counting semaphore.

`#include <nil.h>`

Collaboration diagram for nil_semaphore:



Data Fields

- volatile cnt_t cnt

    Semaphore counter.

6.1.1 Detailed Description

Structure representing a counting semaphore.

6.1.2 Field Documentation

6.1.2.1 volatile cnt_t nil_semaphore::cnt

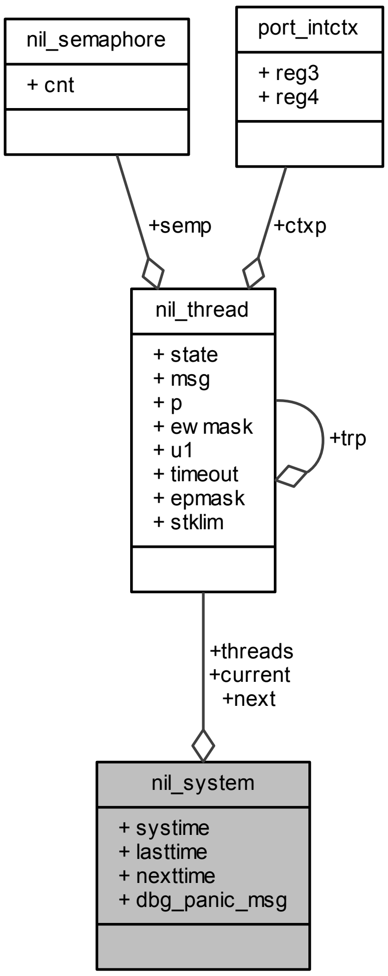
Semaphore counter.

6.2 nil_system Struct Reference

System data structure.

`#include <nil.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.
- thread_t threads [NIL_CFG_NUM_THREADS+1]

  Thread structures for all the defined threads.
- const char ∗volatile dbg_panic_msg

  Panic message.

6.2.1 Detailed Description

System data structure.

Note

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

6.2.2 Field Documentation

6.2.2.1 thread_t∗ nil_system::current

Pointer to the running thread.

6.2.2.2 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.

6.2.2.3 volatile systime_t nil_system::systime

System time.

6.2.2.4 systime_t nil_system::lasttime

System time of the last tick event.

6.2.2.5 systime_t nil_system::nexttime

Time of the next scheduled tick event.

6.2.2.6 thread_t nil_system::threads[NIL_CFG_NUM_THREADS+1]

Thread structures for all the defined threads.

6.2.2.7 const char∗ volatile nil_system::dbg_panic_msg

Panic message.

Note

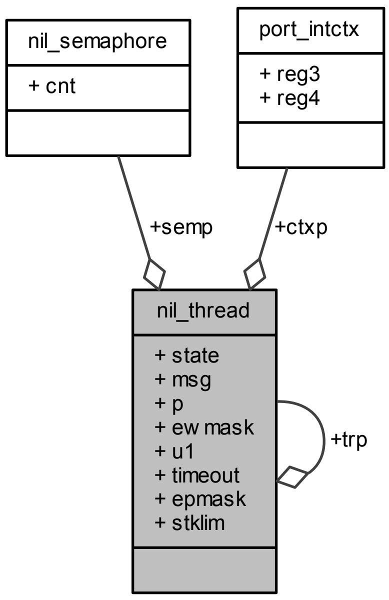
This field is only present if some debug options have been activated.
Accesses to this pointer must never be optimized out so the field itself is declared volatile.

6.3 nil_thread Struct Reference

Structure representing a thread.

`#include <nil.h>`

Collaboration diagram for nil_thread:



Data Fields

- intctx_t ∗ ctxp

  Pointer to internal context.
- tstate_t state

  Thread state.
- volatile systime_t timeout

  Timeout counter, zero if disabled.
- eventmask_t epmask

  Pending events mask.
- stkalign_t ∗ stklim

  Thread stack boundary.
- msg_t msg

  Wake-up message.
- void ∗ p

  Generic pointer.
- thread_reference_t ∗ trp

  Pointer to thread reference.
- semaphore_t ∗ semp

  Pointer to semaphore.
- eventmask_t ewmask

  Enabled events mask.

6.3.1 Detailed Description

Structure representing a thread.

6.3.2 Field Documentation

6.3.2.1 intctx_t∗ nil_thread::ctxp

Pointer to internal context.

6.3.2.2 tstate_t nil_thread::state

Thread state.

6.3.2.3 msg_t nil_thread::msg

Wake-up message.

6.3.2.4 void∗ nil_thread::p

Generic pointer.

6.3.2.5 thread_reference_t∗ nil_thread::trp

Pointer to thread reference.

6.3.2.6 semaphore_t∗ nil_thread::semp

Pointer to semaphore.

6.3.2.7 eventmask_t nil_thread::ewmask

Enabled events mask.

6.3.2.8 volatile systime_t nil_thread::timeout

Timeout counter, zero if disabled.

6.3.2.9 eventmask_t nil_thread::epmask

Pending events mask.

6.3.2.10 stkalign_t∗ nil_thread::stklim

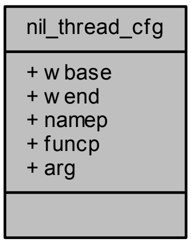
Thread stack boundary.

6.4 nil_thread_cfg Struct Reference

Structure representing a thread static configuration.

```
#include <nil.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.

6.4.1 Detailed Description

Structure representing a thread static configuration.

6.4.2 Field Documentation

6.4.2.1 stkalign_t∗ nil_thread_cfg::wbase

Thread working area base.

6.4.2.2 stkalign_t∗ nil_thread_cfg::wend

Thread working area end.

6.4.2.3 const char∗ nil_thread_cfg::namep

Thread name, for debugging.

6.4.2.4 tfunc_t nil_thread_cfg::funcp

Thread function.

6.4.2.5 void∗ nil_thread_cfg::arg

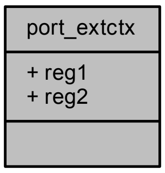
Thread function argument.

6.5 port_extctx Struct Reference

Interrupt saved context.

`#include <nilcore.h>`

Collaboration diagram for port_extctx:



6.5.1 Detailed Description

Interrupt saved context.

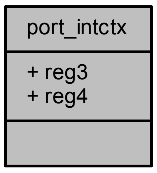
This structure represents the stack frame saved during a preemption-capable interrupt handler.

6.6 port_intctx Struct Reference

System saved context.

`#include <nilcore.h>`

Collaboration diagram for port_intctx:



6.6.1 Detailed Description

System saved context.

This structure represents the inner stack frame during a context switch.

Chapter 7

File Documentation

7.1 nil.c File Reference

Nil RTOS main source file.

```
#include "nil.h"
```

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.
- 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.
- void chSchRescheduleS (void)

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

  Puts the current thread to sleep into the specified state with timeout specification.
- msg_t chThdSuspendTimeoutS (thread_reference_t ∗trp, systime_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 chThdSleep (systime_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 chSemWaitTimeout (semaphore_t ∗sp, systime_t timeout)

  Performs a wait operation on a semaphore with timeout specification.
- msg_t chSemWaitTimeoutS (semaphore_t ∗sp, systime_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, systime_t timeout)

  Waits for any of the specified events.
- eventmask_t chEvtWaitAnyTimeoutS (eventmask_t mask, systime_t timeout)

  Waits for any of the specified events.

Variables

- nil_system_t nil

  System data structures.

7.1.1 Detailed Description

Nil RTOS main source file.

7.2 nil.h File Reference

Nil RTOS main header file.

```
#include "nilconf.h"
#include "niltypes.h"
#include "nilcore.h"
```

Data Structures

- struct nil_semaphore

  Structure representing a counting semaphore.
- 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 NIL_CFG_NUM_THREADS 2

  Number of user threads in the application.
- #define NIL_CFG_ST_RESOLUTION 32

  System time counter resolution.
- #define NIL_CFG_ST_FREQUENCY 100

  System tick frequency.
- #define NIL_CFG_ST_TIMEDELTA 0

  Time delta constant for the tick-less mode.
- #define NIL_CFG_USE_EVENTS TRUE

  Events Flags APIs.
- #define NIL_CFG_ENABLE_ASSERTS FALSE

  System assertions.
- #define NIL_CFG_ENABLE_STACK_CHECK FALSE

  Stack check.
- #define NIL_CFG_SYSTEM_INIT_HOOK() {}

  System initialization hook.
- #define NIL_CFG_THREAD_EXT_FIELDS

  Threads descriptor structure extension.
- #define NIL_CFG_THREAD_EXT_INIT_HOOK(tr) {}

  Threads initialization hook.
- #define NIL_CFG_IDLE_ENTER_HOOK() {}

  Idle thread enter hook.
- #define NIL_CFG_IDLE_LEAVE_HOOK() {}

  Idle thread leave hook.
- #define NIL_CFG_SYSTEM_HALT_HOOK(reason) {}

  System halt hook.

ChibiOS/NIL version identification

- #define CH_KERNEL_VERSION "1.1.3"

  Kernel version string.
- #define CH_KERNEL_MAJOR 1

  Kernel version major number.
- #define CH_KERNEL_MINOR 1

  Kernel version minor number.
- #define CH_KERNEL_PATCH 3

  Kernel version patch number.

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 ((systime_t)-1)

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

  Infinite time specification for all functions with a timeout specification.

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_WTSEM (tstate_t)3

  On semaphore.
- #define NIL_STATE_WTOREVT (tstate_t)4

  Waiting for events.
- #define **NIL_THD_IS_READY**(tr) ((tr)->state == NIL_STATE_READY)
- #define **NIL_THD_IS_SLEEPING**(tr) ((tr)->state == NIL_STATE_SLEEPING)
- #define **NIL_THD_IS_SUSP**(tr) ((tr)->state == NIL_STATE_SUSP)
- #define **NIL_THD_IS_WTSEM**(tr) ((tr)->state == NIL_STATE_WTSEM)
- #define **NIL_THD_IS_WTOREVT**(tr) ((tr)->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[NIL_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.

Working Areas and Alignment

- #define THD_ALIGN_STACK_SIZE(n) ((((n) - 1U) | (sizeof(stkalign_t) - 1U)) + 1U)

  Enforces a correct alignment for a stack area size value.
- #define THD_WORKING_AREA_SIZE(n) THD_ALIGN_STACK_SIZE(PORT_WA_SIZE(n))

  Calculates the total Working Area size.
- #define THD_WORKING_AREA(s, n) stkalign_t s[THD_WORKING_AREA_SIZE(n) / sizeof(stkalign_t)]

  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_PRIORIT←Y(prio)
  Priority level validation macro.
- #define CH_IRQ_PROLOGUE() PORT_IRQ_PROLOGUE()
  IRQ handler enter code.
- #define CH_IRQ_EPILOGUE() PORT_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 S2ST(sec) ((systime_t)((uint32_t)(sec) ∗ (uint32_t)NIL_CFG_ST_FREQUENCY))
  Seconds to system ticks.
- #define MS2ST(msec)
  Milliseconds to system ticks.
- #define US2ST(usec)
  Microseconds to system ticks.

Time conversion utilities for the realtime counter

- #define S2RTC(freq, sec) ((freq) ∗ (sec))
  Seconds to realtime counter.
- #define MS2RTC(freq, msec) (rtcnt_t)((((freq) + 999UL) / 1000UL) ∗ (msec))
  Milliseconds to realtime counter.
- #define US2RTC(freq, usec) (rtcnt_t)((((freq) + 999999UL) / 1000000UL) ∗ (usec))
  Microseconds to realtime counter.

Macro Functions

- #define chSysGetRealtimeCounterX() (rtcnt_t)port_rt_get_counter_value()
  Returns the current value of the system real time counter.
- #define chSysDisable() port_disable()
  Enters the kernel lock mode.
- #define chSysEnable() port_enable()
  Enters the kernel lock mode.
- #define chSysLock() port_lock()
  Enters the kernel lock state.
- #define chSysUnlock() port_unlock()
  Leaves the kernel lock state.
- #define chSysLockFromISR() port_lock_from_isr()
  Enters the kernel lock state from within an interrupt handler.
- #define chSysUnlockFromISR() port_unlock_from_isr()
  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(sec) chThdSleep(S2ST(sec))

Delays the invoking thread for the specified number of seconds.
- #define chThdSleepMilliseconds(msec) chThdSleep(MS2ST(msec))

    Delays the invoking thread for the specified number of milliseconds.
- #define chThdSleepMicroseconds(usec) chThdSleep(US2ST(usec))

    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 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 chSemGetCounterI(sp) ((sp)->cnt)

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

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

    Returns the elapsed time since the specified start time.
- #define chVTIsTimeWithinX(time, start, end) ((bool)((systime_t)((time) - (start)) < (systime_t)((end) - (start))))

    Checks if the specified time is within the specified time window.
- #define chDbgAssert(c, r)

    Condition assertion.

Typedefs

- typedef struct nil_thread thread_t

    Type of a structure representing a thread.
- typedef struct port_intctx intctx_t

    Type of internal context structure.
- typedef struct nil_semaphore 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.
- void chSchRescheduleS (void)

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

    Puts the current thread to sleep into the specified state with timeout specification.
- msg_t chThdSuspendTimeoutS (thread_reference_t ∗trp, systime_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 chThdSleep (systime_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 chSemWaitTimeout (semaphore_t ∗sp, systime_t timeout)

    Performs a wait operation on a semaphore with timeout specification.
- msg_t chSemWaitTimeoutS (semaphore_t ∗sp, systime_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, systime_t timeout)

    Waits for any of the specified events.
- eventmask_t chEvtWaitAnyTimeoutS (eventmask_t mask, systime_t timeout)

    Waits for any of the specified events.

Variables

- stkalign_t __main_thread_stack_base__

7.2.1 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.

7.3 nilconf.h File Reference

Configuration file template.

Macros

Kernel parameters and options

- #define NIL_CFG_NUM_THREADS 1

  Number of user threads in the application.

System timer settings

- #define NIL_CFG_ST_RESOLUTION 32

  System time counter resolution.
- #define NIL_CFG_ST_FREQUENCY 50000

  System tick frequency.
- #define NIL_CFG_ST_TIMEDELTA 2

  Time delta constant for the tick-less mode.

Subsystem options

- #define NIL_CFG_USE_EVENTS TRUE

  Events Flags APIs.

Debug options

- #define NIL_CFG_ENABLE_ASSERTS FALSE

  System assertions.
- #define NIL_CFG_ENABLE_STACK_CHECK FALSE

  Stack check.

Kernel hooks

- #define NIL_CFG_SYSTEM_INIT_HOOK()

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

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

  Threads initialization hook.
- #define NIL_CFG_IDLE_ENTER_HOOK()

  Idle thread enter hook.
- #define NIL_CFG_IDLE_LEAVE_HOOK()

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

  System halt hook.

7.3.1 Detailed Description

Configuration file template.

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

7.4 nilcore.c File Reference

Port code.

```
#include "nil.h"
```

7.4.1 Detailed Description

Port code.

7.5 nilcore.h File Reference

Port macros and structures.

Data Structures

- struct port_extctx

    Interrupt saved context.
- struct port_intctx

    System saved context.

Macros

- #define PORT_INT_REQUIRED_STACK 32

    Per-thread stack overhead for interrupts servicing.
- #define PORT_USE_ALT_TIMER FALSE

    Enables an alternative timer implementation.
- #define PORT_SETUP_CONTEXT(tp, wend, pf, arg)

    Platform dependent thread stack setup.
- #define PORT_WA_SIZE(n)

    Computes the thread working area global size.
- #define PORT_IRQ_IS_VALID_PRIORITY(n) false

    Priority level verification macro.
- #define PORT_IRQ_IS_VALID_KERNEL_PRIORITY(n) false

    Priority level verification macro.
- #define PORT_IRQ_PROLOGUE()

    IRQ prologue code.
- #define PORT_IRQ_EPILOGUE() _port_irq_epilogue()

    IRQ epilogue code.
- #define PORT_IRQ_HANDLER(id) void id(void)

    IRQ handler function declaration.
- #define PORT_FAST_IRQ_HANDLER(id) void id(void)

    Fast IRQ handler function declaration.

- #define port_switch(ntp, otp)

  Performs a context switch between two threads.

Architecture and Compiler

- #define PORT_ARCHITECTURE_XXX

  Macro defining the port architecture.
- #define PORT_ARCHITECTURE_NAME "XXX"

  Name of the implemented architecture.
- #define PORT_CORE_VARIANT_NAME "XXXX-Y"

  Name of the architecture variant.
- #define PORT_COMPILER_NAME "GCC " __VERSION__

  Compiler name and version.
- #define PORT_INFO "port description"

  Port-specific information string.
- #define PORT_SUPPORTS_RT FALSE

  This port supports a realtime counter.

Typedefs

- typedef uint64_t stkalign_t

  Type of stack and memory alignment enforcement.

Functions

- static void port_init (void)

  Port-related initialization code.
- static syssts_t port_get_irq_status (void)

  Returns a word encoding the current interrupts status.
- static bool port_irq_enabled (syssts_t sts)

  Checks the interrupt status.
- static bool port_is_isr_context (void)

  Determines the current execution context.
- static void port_lock (void)

  Kernel-lock action.
- static void port_unlock (void)

  Kernel-unlock action.
- static void port_lock_from_isr (void)

  Kernel-lock action from an interrupt handler.
- static void port_unlock_from_isr (void)

  Kernel-unlock action from an interrupt handler.
- static void port_disable (void)

  Disables all the interrupt sources.
- static void port_suspend (void)

  Disables the interrupt sources below kernel-level priority.
- static void port_enable (void)

  Enables all the interrupt sources.
- static void port_wait_for_interrupt (void)

  Enters an architecture-dependent IRQ-waiting mode.
- static rtcnt_t port_rt_get_counter_value (void)

  Returns the current value of the realtime counter.

7.5.1 Detailed Description

Port macros and structures.

7.6 nilcore_timer.h File Reference

System timer header file.

Functions

- static void port_timer_start_alarm (systime_t abstime)

  Starts the alarm.
- static void port_timer_stop_alarm (void)

  Stops the alarm interrupt.
- static void port_timer_set_alarm (systime_t abstime)

  Sets the alarm time.
- static systime_t port_timer_get_time (void)

  Returns the system time.
- static systime_t port_timer_get_alarm (void)

  Returns the current alarm time.

7.6.1 Detailed Description

System timer header file.

7.7 niltypes.h File Reference

Port system types.

```
#include <stddef.h>
#include <stdint.h>
#include <stdbool.h>
```

Macros

- #define ROMCONST const

  ROM constant modifier.
- #define NOINLINE __attribute__((noinline))

  Makes functions not inlineable.
- #define PORT_THD_FUNCTION(tname, arg) __attribute__((noreturn)) void tname(void ∗arg)

  Optimized thread function declaration macro.
- #define PACKED_VAR __attribute__((packed))

  Packed variable specifier.

Common constants

- #define FALSE 0

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

  Generic 'true' boolean constant.

Typedefs

- typedef uint32_t syssts_t
- typedef uint32_t rtcnt_t
- typedef uint8_t tstate_t
- typedef int32_t msg_t
- typedef uint32_t eventmask_t
- typedef int32_t cnt_t
- typedef uint32_t ucnt_t
- typedef uint32_t systime_t

    Type of system time.

7.7.1 Detailed Description

Port system types.

Index