These files document the internal implementation of eCos which may or may not change in later revisions. The interface presented here has no guarantee of being supported in future releases of eCos.

Copyright (c) 2002 by Navosha Inc.. This material may be distributed only subject to the terms and conditions set forth in the Open Publication License, v1.0 or later (the latest version is presently available at http://www.opencontent.org/openpub/).

Distribution of the work or derivative of the work in any standard (paper) book form is prohibited unless prior permission is obtained from the copyright holder. The holder my be contacted at staff@navosha.com. This document may be reproduced in all or part electronically as well as modified provided that the Navosha logo is incorporated in the resulting document and all copyright notices are included.

NOTE: this is an **ALPHA** document currently. There may be errors as well as omissions in this document. Your feedback is welcome and appreciated.

Table of Contents

Function Index

Cyg_Condition_Variable::wait	wait on a condition variable
Cyg_Condition_Variable::wait	wait until an absolute time on a condition variable
Cyg_Counter::Cyg_Counter	create a counter
Cyg_Counter::~Cyg_Counter	destroys a counter
Cyg_Counter::current_value	get the current value of the counter
Cyg_Counter::current_value_lo	get lower 32 bits of counter
Cyg_Counter::current_value_hi	get upper 32 bits of counter
Cyg_Counter::set_value	set counter value directly
Cyg_Counter::tick	increment counter by some number of ticks
Cyg_Counter::add_alarm	attach an alarm to counter
Cyg_Counter::rem_alarm	detach alarm from counter
Cyg_Counting_Semaphore::Cyg_Counting_Semaphore	create counting semaphore
Cyg_Counting_Semaphore::~Cyg_Counting_Semaphore	destroy a counting semaphore
Cyg_Counting_Semaphore::wait	get a counting semaphore
Cyg_Counting_Semaphore::wait	wait until an absolute time for a counting semaphore
Cyg_Counting_Semaphore::trywait	get a counting semaphore, don't block
Cyg_Counting_Semaphore::post	release a counting semaphore
Cyg_Counting_Semaphore::peek	get the count of a counting semaphore
Cyg_Flag::Cyg_Flag	create flag
Cyg_Flag::~Cyg_Flag	destroy flag
Cyg_Flag::setbits	set bits in a flag
Cyg_Flag::maskbits	clear bits in flag
Cyg_Flag::wait	wait on a a condition or set of conditions
Cyg_Flag::wait	wait on a a condition or set of conditions with a timeout
Cyg_Flag::poll	test for a pattern match on the flag
Cyg_Flag::peek	get conditions set in a given flag
Cyg_Flag::waiting	check to see if any threads are waiting on flag
Cyg_Interrupt::Cyg_Interrupt	create an interrupt handler
Cyg_Interrupt::~Cyg_Interrupt	delete an interrupt handler

Cyg_Interrupt::attach	attach an interrupt
Cyg_Interrupt::detach	detach an interrupt
Cyg_Interrupt::get_vsr	get the VSR pointer of an interrupt
Cyg_Interrupt::set_vsr	set a new VSR
Cyg_Interrupt::disable_interrupts	disable interrupts globally
Cyg_Interrupt::enable_interrupts	enable interrupts globally
Cyg_Interrupt::mask_interrupt	mask an interrupt
Cyg_Interrupt::mask_interrupt_intunsafe	interrupt unsafe mask an interrupt
Cyg_Interrupt::unmask_interrupt	unmask an interrupt
Cyg_Interrupt::unmask_interrupt_intunsafe	interrupt unsafe unmask an interrupt
Cyg_Interrupt::acknowledge_interrupt	acknowledge an interrupt
Cyg_Interrupt::configure_interrupt	configure an interrupt
Cyg_Interrupt::set_cpu	set cpu
Cyg_Interrupt::get_cpu	get cpu
Cyg_Mbox::Cyg_Mbox	create a message box
Cyg_Mbox::~Cyg_Mbox	destroy a message box
Cyg_Mbox::get	get a message from a message box
Cyg_Mbox::get	get a message from a message box with timeout
Cyg_Mbox::tryget	get a message from a message box if one is available
Cyg_Mbox::peek_item	get a message from an mbox but don't remove from queue
Cyg_Mbox::put	place a message into a message box
Cyg_Mbox::put	place a message into a message box with a timeout
Cyg_Mbox::tryput	place a message into a message box if space is available
Cyg_Mbox::peek	gets the number of messages currently in the message queue
Cyg_Mbox::waiting_to_get	reports if any threads are waiting to get a message from this mbox
Cyg_Mbox::waiting_to_put	reports if any threads are waiting to place a message into this mbox
Cyg_Mempool_Fixed::Cyg_Mempool_Fixed	create fixed block memory heap

Cyg_Mempool_Fixed::~Cyg_Mempool_Fixed	destroy fixed size memory pool
Cyg_Mempool_Fixed::try_alloc	allocate a fixed block of memory, nonblocking
Cyg_Mempool_Fixed::alloc	allocate a fixed block size, block thread if necessary
Cyg_Mempool_Fixed::alloc	allocate a fixed block size with timeout
Cyg_Mempool_Fixed::free	return a block of memory to a fixed sized heap
Cyg_Mempool_Fixed::get_status	get status on a heap
Cyg_Mempool_Variable::Cyg_Mempool_Variable	create a variable heap
Cyg_Mempool_Variable::~Cyg_Mempool_Variable	destroy variable heap
Cyg_Mempool_Variable::try_alloc	allocate a block of memory
Cyg_Mempool_Variable::alloc	allocate a block of memory, block until memory available
Cyg_Mempool_Variable::alloc	allocate a block of memory with timeout
Cyg_Mempool_Variable::resize_alloc	resize a previously allocated block of memory
Cyg_Mempool_Variable::free	free an allocated block of memory
Cyg_Mempool_Variable::get_status	get status on a heap
Cyg_Mutex::Cyg_Mutex	create a mutex
Cyg_Mutex::Cyg_Mutex	create a mutex with a specified protocol
Cyg_Mutex::~Cyg_Mutex	destroy a mutex
Cyg_Mutex::lock	lock a mutex or wait until it can be locked
Cyg_Mutex::trylock	lock a mutex if it's free
Cyg_Mutex::unlock	unlock a mutex
Cyg_Mutex::release	release all threads waiting on a mutex
Cyg_Mutex::set_ceiling	set the max priority to be inherited
Cyg_Mutex::get_ceiling	get the priority ceiling of this mutex
Cyg_Mutex::get_owner	get the current owner of a mutex
Cyg_Mutex::set_protocol	set the protocol of a mutex
Cyg_Scheduler::get_sched_lock	get the lock count of a thread
Cyg_Scheduler::lock	lock a thread
Cyg_Scheduler::unlock	unlock a thread

Cyg_Thread::Cyg_Thread	contructor to create a new thread
Cyg_Thread::exit	terminate calling thread
Cyg_Thread::suspend	suspend a thread
Cyg_Thread::resume	resume a suspended thread
Cyg_Thread::force_resume	force a suspended thread to be resumed
Cyg_Thread::kill	kill a thread
Cyg_Thread::release	force a thread to wake up with the reason of BREAK
Cyg_Thread::yield	yield the cpu to another thread
Cyg_Thread::self	get the "this" pointer of the calling thread
Cyg_Thread::set_priority	set priority of a thread
Cyg_Thread::get_priority	get the set priority of a thread
Cyg_Thread::get_current_priority	get the current priority of a thread
Cyg_Thread::delay	delay a thread
Cyg_HardwareThread::get_stack_base	get base address of a thread's stack
Cyg_HardwareThread::get_stack_size	get the size of a thread's stack
Cyg_HardwareThread::measure_stack_usage	measure a stack's usage
Cyg_Thread::new_data_index	gets a new data index for per thread data
Cyg_Thread::free_data_index	free a data index for per thread data
Cyg_Thread::get_data	get per thread data
Cyg_Thread::get_data_ptr	get per thread data pointer
Cyg_Thread::set_data	set per thread data
Cyg_Thread::add_destructor	add a thread destructor
Cyg_Thread::rem_destructor	remove a thread destructor
Cyg_Thread::register_exception	register an exception handler
Cyg_Thread::deregister_exception	deregister an exception

Cyg_Alarm::Cyg_Alarm

Name: *Cyg_Alarm::Cyg_Alarm* **( )** - create an alarm

Synopsis:

```
Cyg_Alarm::Cyg_Alarm
(
  Cyg_Counter  *counter, /* Attached to this counter */
  cyg_alarm_fn *alarm,   /* Call-back function       */
  CYG_ADDRWORD data      /* Call-back data           */
)
```

Description: This creates a new alarm and attaches it to the specified counter. When the alarm expires the call-back function "alarm" will be called. The callback function takes one argument "data".

The callback is of the form: void cyg_alarm_fn(Cyg_Alarm *alarm, CYG_ADDRWORD data).

Include: #include <**cyg/kernel/clock.hxx**>

Returns: nothing

See Also: Cyg_Alarm::~Cyg_Alarm

Cyg_Alarm::~Cyg_Alarm

Name: *Cyg_Alarm::~Cyg_Alarm* **( )** - destroy an alarm

Synopsis:

```
Cyg_Alarm::~Cyg_Alarm
(
  void
)
```

Description: This disables and destroys an alarm.

Include: #include <**cyg/kernel/clock.hxx**>

Returns: nothing

See Also: Cyg_Alarm::Cyg_Alarm

Cyg_Alarm::initialize

Name: *Cyg_Alarm::initialize* **( )** - initialize an alarm

Synopsis:

```
void Cyg_Alarm::initialize
(
  cyg_tick_count trigger,   /* Absolute trigger time       */
  cyg_tick_count interval=0 /* Relative retrigger interval */
)
```

Description: This initializes an alarm. The trigger time is an absolute value of the associated counter. If the interval is set to 0, the alarm will not retrigger. If the interval is non 0, this alarm will reset automatically to fire again at trigger+interval, then trigger+(2*interval), etc.

The alarm will be enabled automatically after this call

Include: #include <**cyg/kernel/clock.hxx**>

Returns: nothing

See Also: Cyg_Alarm::enable, Cyg_Alarm::disable, Cyg_Alarm::get_times

---

Cyg_Alarm::enable

Name: ***Cyg_Alarm::enable* ( )** - enable an alarm

Synopsis:

```
void Cyg_Alarm::enable
(
  void
)
```

Description: This enables an alarm. This is most often used when a periodic alarm has been disabled.

A periodic alarm that has been disabled and later re-enabled will fire at the same intervals it did previously. For example, a periodic alarm that fired every 10 seconds at time T0, T10, T20, T30... etc that was disabled for 15 seconds at time T31 and then renabled would then start firing again at T50, T60, T70 etc.

If this behavior is not desired, use Cyg_Alarm::initialize to reset the intervals.

Include: #include <**cyg/kernel/clock.hxx**>

Returns: nothing

See Also: Cyg_Alarm::initialize, Cyg_Alarm::disable, Cyg_Alarm::get_times

---

Cyg_Alarm::disable

Name: ***Cyg_Alarm::disable* ( )** - disable an alarm

Synopsis:

```
void Cyg_Alarm::disable
(
  void
)
```

Description: Disables an alarm. Most often used to stop a periodic alarm. This can also be used to cancel an alarm, although using the destructor to do that might be more logical.

Cyg_Alarm::initialize or Cyg_Alarm::enable can be used to re-enable the alarm once it's been disabled.

Include: #include <**cyg/kernel/clock.hxx**>
#include <**cyg/kernel/clock.inl**>

Returns: nothing

See Also: Cyg_Alarm::initialize, Cyg_Alarm::enable

Cyg_Alarm::get_times

Name: ***Cyg_Alarm::get_times* ( ) -** get trigger and interval values

Synopsis:

```
void Cyg_Alarm::get_times
(
  cyg_tick_count *trigger, /* pointer to the next trigger time */
  cyg_tick_count *interval /* pointer to the current interval  */
)
```

Description: Get through pointers the next trigger time and the periodic retrigger interval. The function itself returns nothing.

It is legal to pass NULL as the "trigger" and "interval" pointers if those values are not needed.

Include: #include <**cyg/kernel/clock.hxx**>

Returns: nothing

See Also: Cyg_Alarm::initialize

Cyg_Binary_Semaphore::Cyg_Binary_Semaphore

Name: *Cyg_Binary_Semaphore::Cyg_Binary_Semaphore* **( )** - create a binary semaphore

Synopsis:

```
Cyg_Binary_Semaphore::Cyg_Binary_Semaphore
(
  cyg_bool init_state = false  /* initial state */
)
```

Description: Creates a binary semaphore. Posting a binary semaphore that is already available will remain available, it will not affect its count.

An initial state of "true" means the binary semaphore was created as not taken (free to be taken by any thread) and a state of "false" means the binary semaphore was created as being taken.

Include: #include <**cyg/kernel/sema.hxx**>

Returns: nothing

See Also: Cyg_Binary_Semaphore::~Cyg_Binary_Semaphore

Cyg_Binary_Semaphore::~Cyg_Binary_Semaphore

Name: *Cyg_Binary_Semaphore::~Cyg_Binary_Semaphore* **( )** - destroy a binary semaphore

Synopsis:

```
Cyg_Binary_Semaphore::~Cyg_Binary_Semaphore
(
  void
)
```

Description: Destroys a binary semaphore. This will NOT release threads waiting on the binary semaphore, it will simply call the destructor which essentially does nothing but free the memory. Be certain that the semaphore is available before destroying it.

Include: #include <**cyg/kernel/sema.hxx**>

Returns: nothing

See Also: Cyg_Binary_Semaphore::Cyg_Binary_Semaphore

Cyg_Binary_Semaphore::wait

Name: ***Cyg_Binary_Semaphore::wait* ( )** - get a binary semaphore

Synopsis:

```
cyg_bool Cyg_Binary_Semaphore::wait
(
  void
)
```

Description: Takes a binary semaphore. If the binary semaphore is not available this will block until the binary semaphore is available.

Include: #include <**cyg/kernel/sema.hxx**>

Returns: "true" is the binary semaphore was taken, "false" if the binary semaphore could not be taken. The value of "false" will be returned if the thread is awaken. See the thread api.

See Also: Cyg_Binary_Semaphore::trywait, Cyg_Binary_Semaphore::post, Cyg_Binary_Semaphore::posted

---

Cyg_Binary_Semaphore::trywait

Name: ***Cyg_Binary_Semaphore::trywait* ( )** - get a binary semaphore, don't block

Synopsis:

```
cyg_bool Cyg_Binary_Semaphore::trywait
(
  void
)
```

Description: Takes a binary semaphore but only if it is currently available. If the binary semaphore has already been taken by another thread this will return "false".

Include: #include <**cyg/kernel/sema.hxx**>

Returns: "true" if the semaphore was taken by the calling thread, "false" if the semaphore was already taken by a thread.

See Also: Cyg_Binary_Semaphore::wait, Cyg_Binary_Semaphore::post, Cyg_Binary_Semaphore::posted

---

Cyg_Binary_Semaphore::post

Name: ***Cyg_Binary_Semaphore::post* ( )** - release a binary semaphore

Synopsis:

```
void Cyg_Binary_Semaphore::post
(
  void
)
```

Description: This will release a binary semaphore. If the binary semaphore is already released, calling this will have no affect on the binary semaphore. Unlike a mutex, a binary semaphore can be released by any thread, not just the thread that allocated it.

Include: #include <**cyg/kernel/sema.hxx**>

Returns: nothing

See Also: Cyg_Binary_Semaphore::wait, Cyg_Binary_Semaphore::trywait, Cyg_Binary_Semaphore::posted

---

Cyg_Binary_Semaphore::posted

Name: ***Cyg_Binary_Semaphore::posted*** **( )** **-** check availability of a binary semaphore

Synopsis:

```
cyg_bool Cyg_Binary_Semaphore::posted
(
  void
)
```

Description: This reports the status of a binary semaphore.

Include: #include <**cyg/kernel/sema.hxx**>

Returns: "true" if the binary semaphore is available "false" if a thread has allocated it already.

See Also: Cyg_Binary_Semaphore::wait, Cyg_Binary_Semaphore::trywait, Cyg_Binary_Semaphore::post

---

Cyg_Clock::Cyg_Clock

Name: *Cyg_Clock::Cyg_Clock* **( )** - create a clock

Synopsis:

```
Cyg_Clock::Cyg_Clock
(
  cyg_resolution resolution /* resolution */
)
```

Description: Creates a clock with an associated resolution.

All a clock is is a counter with an associated resolution. It's a class derived from Cyg_Counter. All you use it for is to store and save resolutions of the clock.

Since this is a derived class, study the "counter" documentation for more information on counters.

Include: #include <**cyg/kernel/clock.hxx**>

Returns: nothing

See Also: Cyg_Clock::~Cyg_Clock

Cyg_Clock::~Cyg_Clock

Name: *Cyg_Clock::~Cyg_Clock* **( )** - destroy a clock

Synopsis:

```
Cyg_Clock::~Cyg_Clock
(
  void
)
```

Description: This destroys a clock and the counter it's associated with

Include: #include <**cyg/kernel/clock.hxx**>

Returns: nothing

See Also: Cyg_Clock::Cyg_Clock

Cyg_Clock::get_resolution

Name: *Cyg_Clock::get_resolution* **( )** - get the resolution of the clock

Synopsis:

```
cyg_resolution Cyg_Clock::get_resolution
(
  void
)
```

Description: Gets the resolution of this clock

Include: #include <**cyg/kernel/clock.hxx**>
#include <**cyg/kernel/clock.inl**>

Returns: the resolution of this clock.

See Also: Cyg_Clock::set_resolution

Cyg_Clock::set_resolution

Name: *Cyg_Clock::set_resolution* **( )** - set clock resolution

Synopsis:

```
void Cyg_Clock::set_resolution
(
  cyg_resolution resolution /* new resolution */
)
```

Description: This sets the resolution of the clock

Include: #include <**cyg/kernel/clock.hxx**>
#include <**cyg/kernel/clock.inl**>

Returns: nothing

See Also: Cyg_Clock::get_resolution

Cyg_Clock::get_other_to_clock_converter

Name: *Cyg_Clock::get_other_to_clock_converter* **( )** - setup clock conversion

Synopsis:

```
void Cyg_Clock::get_other_to_clock_converter
(
  cyg_uint64       ns_per_other_tick, /* ns/tick of the other clock */
  struct converter *pcc               /* conversion struct          */
)
```

Description: THIS DESCRIPTION MAY BE WRONG

This, as far as I can tell, sets up a conversion from one clock to another. This must be used in conjunction with Cyg_Clock::convert to be of any use. This is so you can relate one clock (or counter) to another.

This converts from the other clock to this clock.

the struct converter is: struct converter {cyg_uint64 mul1, div1, mul2, div2;}. Clock ticks = (((otherticks*mul1)/div1)*mul2/div2)

Include: #include <**cyg/kernel/clock.hxx**>

Returns: nothing

See Also: Cyg_Clock::get_clock_to_other_converter, Cyg_Clock::convert

---

Cyg_Clock::get_clock_to_other_converter

Name: ***Cyg_Clock::get_clock_to_other_converter*** **( )** - setup clock conversion

Synopsis:

```
void Cyg_Clock::get_clock_to_other_converter
(
  cyg_uint64       ns_per_other_tick, /* ns/tick of the other clock */
  struct converter *pcc               /* conversion struct          */
)
```

Description: THIS DESCRIPTION MAY BE WRONG

This, as far as I can tell, sets up a conversion from one clock to another. This must be used in conjunction with Cyg_Clock::convert to be of any use. This is so you can relate one clock (or counter) to another.

This converts from this clock to the other clock.

the struct converter is: struct converter {cyg_uint64 mul1, div1, mul2, div2;}. Clock ticks = (((otherticks*mul1)/div1)*mul2/div2)

Include: #include <**cyg/kernel/clock.hxx**>

Returns: nothing

See Also: Cyg_Clock::get_other_to_clock_converter, Cyg_Clock::convert

---

Cyg_Clock::convert

Name: ***Cyg_Clock::convert*** **( )** - convert from one clock to another

Synopsis:

```
static cyg_tick_count Cyg_Clock::convert
(
  cyg_tick_count   value, /* ns/tick of this clock */
  struct converter *pcc   /* conversion struct     */
)
```

Description: THIS DESCRIPTION MAY BE WRONG

I think this does a conversion from this clock value to another. I need help with this. It's not obvious why these are broken up into 3 seperate function calls. I don't see many people needing this functionality.

Include: #include <**cyg/kernel/clock.hxx**>
#include <**cyg/kernel/clock.inl**>

Returns: The converted clock ticks.

See Also: Cyg_Clock::get_other_to_clock_converter, Cyg_Clock::get_clock_to_other_converter

---

Cyg_Condition_Variable::Cyg_Condition_Variable

Name: ***Cyg_Condition_Variable::Cyg_Condition_Variable*** **( )** - create a condition variable

Synopsis:

```
Cyg_Condition_Variable::Cyg_Condition_Variable
(
  Cyg_Mutex &mutex /* associated mutex */
)
```

Description: Creates a condition variable.

Include: #include <**cyg/kernel/mutex.hxx**>

Returns: nothing

See Also: Cyg_Condition_Variable::~Cyg_Condition_Variable

Cyg_Condition_Variable::~Cyg_Condition_Variable

Name: ***Cyg_Condition_Variable::~Cyg_Condition_Variable*** **( )** - destroy condition variable

Synopsis:

```
Cyg_Condition_Variable::~Cyg_Condition_Variable
(
  void
)
```

Description: This destroys a condition variable. Be careful not do destroy a condition variable that is currently in use. The mutex associated with the condition variable needs to be destroyed seperately.

Include: #include <**cyg/kernel/mutex.hxx**>

Returns: nothing

See Also: Cyg_Condition_Variable::Cyg_Condition_Variable

Cyg_Condition_Variable::signal

Name: ***Cyg_Condition_Variable::signal*** **( )** - wake one thread waiting on condition variable

Synopsis:
```
void Cyg_Condition_Variable::signal
(
  void
)
```

Description: Wakes a thread waiting on a condition variable. If multiple threads are waiting on the thread, which one wakes up depends on which scheduler is being used. The mlqueue scheduler is the most often, this would mean that the thread with the highest priority will wake.

Include: #include <**cyg/kernel/mutex.hxx**>

Returns: nothing

See Also: Cyg_Condition_Variable::broadcast, Cyg_Condition_Variable::wait

---

Cyg_Condition_Variable::broadcast

Name: ***Cyg_Condition_Variable::broadcast* ( )** - wake all threads waiting on condition variable

Synopsis:
```
void Cyg_Condition_Variable::broadcast
(
  void
)
```

Description: This wakes all threads waiting on the condition variable.

Include: #include <**cyg/kernel/mutex.hxx**>

Returns: nothing

See Also: Cyg_Condition_Variable::signal, Cyg_Condition_Variable::wait

---

Cyg_Condition_Variable::wait

Name: ***Cyg_Condition_Variable::wait* ( )** - wait on a condition variable

Synopsis:
```
cyg_bool Cyg_Condition_Variable::wait
(
  void
)
```

Description: This causes the calling thread to wait on a condition variable. It will wait forever.

Include: #include <**cyg/kernel/mutex.hxx**>

Returns: "true" if the thread was awakened normally, "false" if the thread was awakened by a Cyg_Thread::BREAK or Cyg_Thread::DESTRUCT signal.

See Also: Cyg_Condition_Variable::signal, Cyg_Condition_Variable::broadcast

---

Cyg_Condition_Variable::wait

Name: ***Cyg_Condition_Variable::wait* ( )** - wait until an absolute time on a condition variable

Synopsis:

```
cyg_bool Cyg_Condition_Variable::wait
(
  cyg_tick_count absolute_time /* absolute time */
)
```

Description: This causes the calling thread to wait on a condition variable for until the system reaches the absolute time specified or until the condition variable is available, whichever comes first.

Include: #include <**cyg/kernel/mutex.hxx**>

Returns: "true" if the thread was awakened normally, "false" if the thread timed out waiting on the condition variable or if the thread was awakened by a Cyg_Thread::BREAK or Cyg_Thread::DESTRUCT signal.

See Also: Cyg_Condition_Variable::signal, Cyg_Condition_Variable::broadcast

---

Cyg_Condition_Variable::wait

Name: ***Cyg_Condition_Variable::wait* ( )** - wait on a condition variable

Synopsis:

```
cyg_bool Cyg_Condition_Variable::wait
(
  Cyg_Mutex &mx  /* other mutex to use */
)
```

Description: This causes the calling thread to wait on a condition variable but uses another mutex instead of the mutex that was associated with this condition variable on creation.

It is probably not a good idea to use this

Include: #include <**cyg/kernel/mutex.hxx**>

Returns: "true" if the thread was awakened normally, "false" if the thread was awakened by a Cyg_Thread::BREAK or Cyg_Thread::DESTRUCT signal.

See Also: Cyg_Condition_Variable::signal, Cyg_Condition_Variable::broadcast

---

Cyg_Condition_Variable::wait

Name: ***Cyg_Condition_Variable::wait* ( )** - wait until an absolute time on a condition variable

Synopsis:

```
cyg_bool Cyg_Condition_Variable::wait
(
  Cyg_Mutex      &mx,          /* other mutex to use */
  cyg_tick_count absolute_time /* absolute time      */
)
```

Description: This causes the calling thread to wait on a condition variable for until the system reaches the absolute time specified or until the condition variable is available, whichever comes first. This uses another mutex instead of the mutex that was associated with this condition variable on creation.

It is probably not a good idea to use this

Include: #include <**cyg/kernel/mutex.hxx**>

Returns: "true" if the thread was awakened normally, "false" if the thread timed out waiting on the condition variable or if the thread was awakened by a Cyg_Thread::BREAK or Cyg_Thread::DESTRUCT signal.

See Also: Cyg_Condition_Variable::signal, Cyg_Condition_Variable::broadcast

---

Cyg_Counter::Cyg_Counter

Name: *Cyg_Counter::Cyg_Counter* **( )** - create a counter

Synopsis:

```
Cyg_Counter::Cyg_Counter
(
  cyg_uint32 increment=1 /* number of ticks to increment counter by */
)
```

Description: This creates a new counter. Counters can increment any arbitrary amount but usually only by 1. The value of the counter is always initialized to 0. Counters are 64 bit.

Include: #include <**cyg/kernel/clock.hxx**>

Returns: nothing

See Also: Cyg_Counter::~Cyg_Counter

Cyg_Counter::~Cyg_Counter

Name: *Cyg_Counter::~Cyg_Counter* **( )** - destroys a counter

Synopsis:

```
Cyg_Counter::~Cyg_Counter
(
  void
)
```

Description: This destroys a counter.

Include: #include <**cyg/kernel/clock.hxx**>

Returns: nothing

See Also: Cyg_Counter::Cyg_Counter

Cyg_Counter::current_value

Name: *Cyg_Counter::current_value* **( )** - get the current value of the counter

Synopsis:

```
cyg_tick_count Cyg_Counter::current_value
(
  void
)
```

Description: Gets the current value of the counter.

Include: #include <**cyg/kernel/clock.hxx**>
#include <**cyg/kernel/clock.inl**>

Returns: the current value of the counter.

See Also: Cyg_Counter::current_value_lo, Cyg_Counter::current_value_hi, Cyg_Counter::set_value, Cyg_Counter::tick

---

Cyg_Counter::current_value_lo

Name: ***Cyg_Counter::current_value_lo* ( )** - get lower 32 bits of counter

Synopsis:

```
cyg_uint32 Cyg_Counter::current_value_lo
(
  void
)
```

Description: Gets the lower 32 bits of the current value of the counter.

Include: #include <**cyg/kernel/clock.hxx**>
#include <**cyg/kernel/clock.inl**>

Returns: the lower 32 bits of the current value of the counter.

See Also: Cyg_Counter::current_value, Cyg_Counter::current_value_hi, Cyg_Counter::set_value, Cyg_Counter::tick

---

Cyg_Counter::current_value_hi

Name: ***Cyg_Counter::current_value_hi* ( )** - get upper 32 bits of counter

Synopsis:

```
cyg_uint32 Cyg_Counter::current_value_hi
(
  void
)
```

Description: Gets the upper 32 bits of the current value of the counter.

Include: #include <**cyg/kernel/clock.hxx**>
#include <**cyg/kernel/clock.inl**>

Returns: the upper 32 bits of the current value of the counter.

See Also: Cyg_Counter::current_value, Cyg_Counter::current_value_lo, Cyg_Counter::set_value, Cyg_Counter::tick

---

Cyg_Counter::set_value

Name: ***Cyg_Counter::set_value* ( )** - set counter value directly

Synopsis:

```
void Cyg_Counter::set_value
(
  cyg_tick_count new_value /* new value of counter */
)
```

Description: Sets the value of the counter. Note that the value is a 64 bit value not a 32 bit value.

Include: #include <**cyg/kernel/clock.hxx**>
#include <**cyg/kernel/clock.inl**>

Returns: nothing

See Also: Cyg_Counter::current_value, Cyg_Counter::current_value_lo, Cyg_Counter::current_value_hi, Cyg_Counter::tick

---

Cyg_Counter::tick

Name: ***Cyg_Counter::tick* ( )** - increment counter by some number of ticks

Synopsis:

```
void Cyg_Counter::tick
(
  cyg_uint32 ticks=1 /* number of ticks to increment counter */
)
```

Description: Increments the counter by some number of ticks. Doing this can trigger alarms that are attached to this counter if the alarms have expired.

Include: #include <**cyg/kernel/clock.hxx**>

Returns: nothing

See Also: Cyg_Counter::current_value, Cyg_Counter::current_value_lo, Cyg_Counter::current_value_hi, Cyg_Counter::set_value

---

Cyg_Counter::add_alarm

Name: ***Cyg_Counter::add_alarm* ( )** - attach an alarm to counter

Synopsis:

```
void Cyg_Counter::add_alarm
(
  Cyg_Alarm *alarm  /* alarm to attach to counter */
)
```

Description: This attaches an alarm to the counter.

Include: #include <**cyg/kernel/clock.hxx**>

Returns: nothing

See Also: Cyg_Counter::tick, Cyg_Counter::rem_alarm

Cyg_Counter::rem_alarm

Name: ***Cyg_Counter::rem_alarm*** **( )** - detach alarm from counter

Synopsis:

```
void Cyg_Counter::rem_alarm
(
  Cyg_Alarm *alarm /* alarm to remove from counter */
)
```

Description: This detaches an alarm from a counter

Include: #include <**cyg/kernel/clock.hxx**>

Returns: nothing

See Also: Cyg_Counter::tick, Cyg_Counter::add_alarm

Cyg_Counting_Semaphore::Cyg_Counting_Semaphore

Name: *Cyg_Counting_Semaphore::Cyg_Counting_Semaphore* **( )** - create counting semaphore

Synopsis:

```
Cyg_Counting_Semaphore::Cyg_Counting_Semaphore
(
  cyg_count32 init_count = 0 /* inital count */
)
```

Description: This creates a counting semaphore.

Include: #include <**cyg/kernel/sema.hxx**>

Returns: nothing

See Also: Cyg_Counting_Semaphore::~Cyg_Counting_Semaphore

Cyg_Counting_Semaphore::~Cyg_Counting_Semaphore

Name: *Cyg_Counting_Semaphore::~Cyg_Counting_Semaphore* **( )** - destroy a counting semaphore

Synopsis:

```
Cyg_Counting_Semaphore::~Cyg_Counting_Semaphore
(
  void
)
```

Description: This destroys a counting semaphore. This will NOT release threads waiting on the counting semaphore, it will simply call the destructor which essentially does nothing but free the memory. Be certain that not threads are waiting on this semaphore before destroying.

Include: #include <**cyg/kernel/sema.hxx**>

Returns: nothing

See Also: Cyg_Counting_Semaphore::Cyg_Counting_Semaphore

Cyg_Counting_Semaphore::wait

Name: *Cyg_Counting_Semaphore::wait* **( )** - get a counting semaphore

Synopsis:

```
cyg_bool Cyg_Counting_Semaphore::wait
(
  void
)
```

Description: Takes a counting semaphore. If the counting semaphore is not available this will block until the counting semaphore is available.

Include: #include <**cyg/kernel/sema.hxx**>

Returns: "true" is the counting semaphore was taken, "false" if the counting semaphore could not be taken. The value of "false" will be returned if the thread is awaken. See the thread api.

See Also: Cyg_Counting_Semaphore::trywait, Cyg_Counting_Semaphore::post, Cyg_Counting_Semaphore::peek

---

Cyg_Counting_Semaphore::wait

Name: ***Cyg_Counting_Semaphore::wait*** **( )** - wait until an absolute time for a counting semaphore

Synopsis:

```
cyg_bool Cyg_Counting_Semaphore::wait
(
  cyg_tick_count timeout /* absolute timeout */
)
```

Description: This grabs a counting semaphore. If the semaphore is not available it will block the allocating thread until "timeout". Timeout is an absolute value, not a relative value. It's a 64 value. You can get the value of the real time clock with Cyg_Clock::real_time_clock->current_value().

Include: #include <**cyg/kernel/sema.hxx**>

Returns: "true" if the semaphore was allocated, "false" is the thread was awaken for another reason. The value of "false" will be returned if the thread is awaken. See the thread api.

See Also: Cyg_Counting_Semaphore::trywait, Cyg_Counting_Semaphore::post, Cyg_Counting_Semaphore::peek

---

Cyg_Counting_Semaphore::trywait

Name: ***Cyg_Counting_Semaphore::trywait*** **( )** - get a counting semaphore, don't block

Synopsis:

```
cyg_bool Cyg_Counting_Semaphore::trywait
(
  void
)
```

Description: Takes a counting semaphore but only if it is currently available. If the counting semaphore has already been taken by another thread this will return "false".

Include: #include <**cyg/kernel/sema.hxx**>

Returns: "true" if the semaphore was taken "false" if it wasn't.

See Also: Cyg_Counting_Semaphore::wait, Cyg_Counting_Semaphore::post, Cyg_Counting_Semaphore::peek

---

Cyg_Counting_Semaphore::post

Name: *Cyg_Counting_Semaphore::post* **( )** - release a counting semaphore

Synopsis:

```
void Cyg_Counting_Semaphore::post
(
  void
)
```

Description: This will increment the count of a counting semaphore so it may be allocated by other threads.

Include: #include <**cyg/kernel/sema.hxx**>

Returns: nothing

See Also: Cyg_Counting_Semaphore::wait, Cyg_Counting_Semaphore::trywait, Cyg_Counting_Semaphore::peek

---

Cyg_Counting_Semaphore::peek

Name: *Cyg_Counting_Semaphore::peek* **( )** - get the count of a counting semaphore

Synopsis:

```
cyg_count32 Cyg_Counting_Semaphore::peek
(
  void
)
```

Description: This just gets the count of a counting semaphore. If any threads are waiting on the counting semaphore, this will of course return 0.

Include: #include <**cyg/kernel/sema.hxx**>

Returns: the count of the counting semaphore.

See Also: Cyg_Counting_Semaphore::wait, Cyg_Counting_Semaphore::trywait, Cyg_Counting_Semaphore::post

---

Cyg_Flag::Cyg_Flag

Name: ***Cyg_Flag::Cyg_Flag* ( )** - create flag

Synopsis:

```
Cyg_Flag::Cyg_Flag
(
  Cyg_FlagValue init=0 /* intial conditions */
)
```

Description: This creates a flag. Flags allow a thread to wait on a condition or a set of conditions. Each condition is represented by a bit. You can control which conditions are set on creation with "init".

Include: #include <**cyg/kernel/flag.hxx**>

Returns: nothing

See Also: Cyg_Flag::~Cyg_Flag

Cyg_Flag::~Cyg_Flag

Name: ***Cyg_Flag::~Cyg_Flag* ( )** - destroy flag

Synopsis:

```
Cyg_Flag::~Cyg_Flag
(
  void
)
```

Description: Destroy a flag. Be careful not to destroy a flag still in use. If you destroy a flag with threads waiting on it, they will be awaken with a signal of Cyg_Thread::DESTRUCT, but this is bad programming practice. Avoid doing this.

Include: #include <**cyg/kernel/flag.hxx**>

Returns: nothing

See Also: Cyg_Flag::Cyg_Flag

Cyg_Flag::setbits

Name: ***Cyg_Flag::setbits*** **( )** - set bits in a flag

Synopsis:

```
void Cyg_Flag::setbits
(
  Cyg_FlagValue arg=~0 /* bits (conditions) to set */
);
```

Description: Sets bits (conditions) in the flag. Each bit represents a single condition.

Include: #include <**cyg/kernel/flag.hxx**>

Returns: nothing

See Also: Cyg_Flag::maskbits, Cyg_Flag::wait, Cyg_Flag::poll, Cyg_Flag::peek, Cyg_Flag::waiting

---

Cyg_Flag::maskbits

Name: ***Cyg_Flag::maskbits*** **( )** - clear bits in flag

Synopsis:

```
void Cyg_Flag::maskbits
(
  Cyg_FlagValue arg=0 /* bits (conditions) to clear */
)
```

Description: This clears bit (conditions) of the flag. Any bit that is set to 1 will not be cleared.

Include: #include <**cyg/kernel/flag.hxx**>

Returns: nothing

See Also: Cyg_Flag::setbits, Cyg_Flag::wait, Cyg_Flag::poll, Cyg_Flag::peek, Cyg_Flag::waiting

---

Cyg_Flag::wait

Name: ***Cyg_Flag::wait*** **( )** - wait on a a condition or set of conditions

Synopsis:

```
Cyg_FlagValue Cyg_Flag::wait
(
  Cyg_FlagValue pattern, /* bit pattern */
  WaitMode      mode     /* mode        */
)
```

Description: This waits on a set of conditions to be set. Once the conditions are met, it will wake the waiting thread. There are several modes that will affect which conditions must be set in order for the thread to wake up. They are:

Cyg_Flag::AND - wait for all conditions to be set in the pattern before waking the thread.

Cyg_Flag::OR - wait for any conditions in the pattern to be set before waking the thread.

Cyg_Flag::CLR - automatically clear the conditions that caused the calling thread to wake.

Cyg_Flag::MASK - clears all conditions set in the pattern

Include: #include <**cyg/kernel/flag.hxx**>

Returns: the flag value that succedded in waking the thread or 0 if there was an error such as a bad pattern or mode.

See Also: Cyg_Flag::setbits, Cyg_Flag::maskbits, Cyg_Flag::poll, Cyg_Flag::peek, Cyg_Flag::waiting

---

Cyg_Flag::wait

Name: ***Cyg_Flag::wait* ( )** - wait on a a condition or set of conditions with a timeout

Synopsis:

```
Cyg_FlagValue Cyg_Flag::wait
(
  Cyg_FlagValue  pattern,    /* bit pattern      */
  WaitMode       mode,       /* mode             */
  cyg_tick_count abs_timeout /* absolute timeout */
)
```

Description: This waits on a set of conditions to be set or times out. Note that the timeout is in absolute, not relative time. Once the conditions are met or the wait expires, it will wake the waiting thread. There are several modes that will affect which conditions must be set in order for the thread to wake up. They are:

Cyg_Flag::AND - wait for all conditions to be set in the pattern before waking the thread.

Cyg_Flag::OR - wait for any conditions in the pattern to be set before waking the thread.

Cyg_Flag::CLR - automatically clear the conditions that caused the calling thread to wake.

Cyg_Flag::MASK - clears all conditions set in the pattern

Include: #include <**cyg/kernel/flag.hxx**>

Returns: the flag value that succedded in waking the thread or 0 if there was an error such as a bad pattern, bad mode, or timeout.

See Also: Cyg_Flag::setbits, Cyg_Flag::maskbits, Cyg_Flag::poll, Cyg_Flag::peek, Cyg_Flag::waiting

---

Cyg_Flag::poll

Name: ***Cyg_Flag::poll* ( )** - test for a pattern match on the flag

Synopsis:

```
Cyg_FlagValue Cyg_Flag::poll
(
  Cyg_FlagValue pattern, /* bit pattern */
  WaitMode      mode     /* mode        */
)
```

Description: This checks a flag for the set of conditions but does not block. The possible modes are:

Cyg_Flag::AND - return match if all conditions in the pattern are set in the flag

Cyg_Flag::OR - return match if any of the conditions in the pattern are set in the flag.

Cyg_Flag::CLR - automatically clear the conditions that caused the calling thread to return a match, IF there was a match.

Cyg_Flag::MASK - clears all conditions set in the pattern if there was a match.

Include: #include <**cyg/kernel/flag.hxx**>

Returns: The pattern that caused the match or 0 if there was no match.

See Also: Cyg_Flag::setbits, Cyg_Flag::maskbits, Cyg_Flag::wait, Cyg_Flag::peek, Cyg_Flag::waiting

---

Cyg_Flag::peek

Name: ***Cyg_Flag::peek* ( )** - get conditions set in a given flag

Synopsis:

```
Cyg_FlagValue Cyg_Flag::peek
(
  void
)
```

Description: This returns the conditions that are currently set in a flag.

Include: #include <**cyg/kernel/flag.hxx**>

Returns: the conditions that are set in a flag as a bitmask.

See Also: Cyg_Flag::setbits, Cyg_Flag::maskbits, Cyg_Flag::wait, Cyg_Flag::poll, Cyg_Flag::waiting

---

Cyg_Flag::waiting

Name: ***Cyg_Flag::waiting* ( )** - check to see if any threads are waiting on flag

Synopsis:

```
cyg_bool Cyg_Flag::waiting
(
  void
)
```

Description: This is used to check to see if any threads are currently waiting on the flag.

Include: #include <**cyg/kernel/flag.hxx**>

Returns: "true" if thread are waiting on the flag, "false" is there are not threads waiting on the flag.

See Also: Cyg_Flag::setbits, Cyg_Flag::maskbits, Cyg_Flag::wait, Cyg_Flag::poll, Cyg_Flag::peek

---

Cyg_Interrupt::Cyg_Interrupt

Name: ***Cyg_Interrupt::Cyg_Interrupt* ( )** - create an interrupt handler

Synopsis:

```
Cyg_Interrupt::Cyg_Interrupt
(
  cyg_vector   vector,   /* Interrupt vector          */
  cyg_priority priority, /* queue priority            */
  CYG_ADDRWORD data,     /* data pointer              */
  cyg_ISR      *isr,     /* interrupt service routine */
  cyg_DSR      *dsr      /* deferred service routine  */
)
```

Description: Constructor for an interrupt handler. The queue priority is used only in the case that interrupts are chained.

The isr has the prototype of cyg_uint32 cyg_ISR(cyg_vector vector, CYG_ADDRWORD data). The ISR is called from the VSR. The VSR is usually implemented by eCos itself. If the ISR returns Cyg_Interrupt::HANDLED the VSR will NOT be called, if the ISR returns Cyg_Interrupt::CALL_DSR the VSR is called.

The vsr has the prototype of void cyg_DSR(cyg_vector vector, cyg_ucount32 count, CYG_ADDRWORD data). The DSR returns nothing.

ISR's cannot access the vast majority of kernel routines. The VSR can access more routines. What can and cannot be called safely from these routines I have not found in the documentation yet.

Include: #include <**cyg/kernel/intr.hxx**>

Returns: nothing

See Also: Cyg_Interrupt::~Cyg_Interrupt

Cyg_Interrupt::~Cyg_Interrupt

Name: ***Cyg_Interrupt::~Cyg_Interrupt* ( )** - delete an interrupt handler

Synopsis:

```
Cyg_Interrupt::~Cyg_Interrupt
(
  void
)
```

Description: This is a destructor for an interrupt handler.

Include:	#include <**cyg/kernel/intr.hxx**>
Returns:	nothing
See Also:	Cyg_Interrupt::Cyg_Interrupt

Cyg_Interrupt::attach

Name: ***Cyg_Interrupt::attach*** **( )** - attach an interrupt

Synopsis:

```
void Cyg_Interrupt::attach
(
  void
)
```

Description: This attaches the ISR and DSR of an interrupt to the physical interrupt. An interrupt must be attached before the ISR or DSR will be used.

Include: #include <**cyg/kernel/intr.hxx**>

Returns: nothing

See Also: Cyg_Interrupt::detach

Cyg_Interrupt::detach

Name: ***Cyg_Interrupt::detach*** **( )** - detach an interrupt

Synopsis:

```
void Cyg_Interrupt::detach
(
  void
)
```

Description: This detaches the ISR and DSR of an interrupt to the physical interrupt.

Include: #include <**cyg/kernel/intr.hxx**>

Returns: nothing

See Also: Cyg_Interrupt::attach

Cyg_Interrupt::get_vsr

Name: ***Cyg_Interrupt::get_vsr*** **( )** - get the VSR pointer of an interrupt

Synopsis:

```
void Cyg_Interrupt::get_vsr
(
  cyg_vector vector, /* interrupt vector                    */
  cyg_VSR    **vsr   /* address of pointer to retrieve VSR */
)
```

Description: This function gets the address of the interrupt's VSR writing into to address pointed to by *vsr.

Include: #include <**cyg/kernel/intr.hxx**>

Returns: nothing

See Also: Cyg_Interrupt::set_vsr

Cyg_Interrupt::set_vsr

Name: ***Cyg_Interrupt::set_vsr* ( ) -** set a new VSR

Synopsis:

```
void Cyg_Interrupt::set_vsr
(
  cyg_vector vector, /* interrupt vector                        */
  cyg_VSR    *vsr,   /* address of the new VSR                  */
  cyg_VSR    **old   /* address of pointer to retrieve old VSR */
)
```

Description: This function sets a new vsr for the interrupt and writes the address of the old vsr to the address pointed to by *old.

Include: #include <**cyg/kernel/intr.hxx**>

Returns: nothing

See Also: Cyg_Interrupt::get_vsr

Cyg_Interrupt::disable_interrupts

Name: ***Cyg_Interrupt::disable_interrupts* ( ) -** disable interrupts globally

Synopsis:

```
void Cyg_Interrupt::disable_interrupts
(
  void
)
```

Description: This disables the interrupts on the calling CPU. Every call to this function must have a matching call to Cyg_Interrupt::enable_interrupts to re-enable interrupts again.

Include: #include <**cyg/kernel/intr.hxx**>

Returns: nothing

See Also: Cyg_Interrupt::enable_interrupts, Cyg_Interrupt::mask_interrupt

Cyg_Interrupt::enable_interrupts

Name: *Cyg_Interrupt::enable_interrupts* **( ) -** enable interrupts globally

Synopsis:

```
void Cyg_Interrupt::enable_interrupts
(
  void
)
```

Description: This enables the interrupts on the calling CPU. If Cyg_Interrupt::disable_interrupts has been called multiple times, this function will have to be called multiple times to actually enable interrupts again.

Include: #include <**cyg/kernel/intr.hxx**>

Returns: nothing

See Also: Cyg_Interrupt::disable_interrupts, Cyg_Interrupt::unmask_interrupt

Cyg_Interrupt::mask_interrupt

Name: *Cyg_Interrupt::mask_interrupt* **( ) -** mask an interrupt

Synopsis:

```
void Cyg_Interrupt::mask_interrupt
(
  cyg_vector vector /* interrupt vector */
)
```

Description: This function masks an interrupt.

Include: #include <**cyg/kernel/intr.hxx**>

Returns: nothing

See Also: Cyg_Interrupt::mask_interrupt_intunsafe, Cyg_Interrupt::unmask_interrupt, Cyg_Interrupt::unmask_interrupt_intunsafe

Cyg_Interrupt::mask_interrupt_intunsafe

Name: *Cyg_Interrupt::mask_interrupt_intunsafe* **( ) -** interrupt unsafe mask an interrupt

Synopsis:

```
void Cyg_Interrupt::mask_interrupt_intunsafe
(
  cyg_vector vector /* interrupt vector */
)
```

Description: This function masks an interrupt but it's not interrupt safe.

Include: #include <**cyg/kernel/intr.hxx**>

Returns: nothing

See Also: Cyg_Interrupt::mask_interrupt, Cyg_Interrupt::unmask_interrupt, Cyg_Interrupt::unmask_interrupt_intunsafe

---

Cyg_Interrupt::unmask_interrupt

Name: ***Cyg_Interrupt::unmask_interrupt*** **( )** - unmask an interrupt

Synopsis:

```
void Cyg_Interrupt::unmask_interrupt
(
  cyg_vector vector /* interrupt vector */
)
```

Description: This function unmasks an interrupt.

Include: #include <**cyg/kernel/intr.hxx**>

Returns: nothing

See Also: Cyg_Interrupt::mask_interrupt, Cyg_Interrupt::mask_interrupt_intunsafe, Cyg_Interrupt::unmask_interrupt_intunsafe

---

Cyg_Interrupt::unmask_interrupt_intunsafe

Name: ***Cyg_Interrupt::unmask_interrupt_intunsafe*** **( )** - interrupt unsafe unmask an interrupt

Synopsis:

```
void Cyg_Interrupt::unmask_interrupt
(
  cyg_vector vector /* interrupt vector */
)
```

Description: This function unmasks an interrupt but it's not interrupt safe.

Include: #include <**cyg/kernel/intr.hxx**>

Returns: nothing

See Also: Cyg_Interrupt::mask_interrupt, Cyg_Interrupt::mask_interrupt_intunsafe, Cyg_Interrupt::unmask_interrupt

---

Cyg_Interrupt::acknowledge_interrupt

Name: ***Cyg_Interrupt::acknowledge_interrupt*** **( )** - acknowledge an interrupt

Synopsis:

```
void Cyg_Interrupt::acknowledge_interrupt
(
  cyg_vector vector /* interrupt vector */
)
```

Description: This function acknowledges an interrupt.

Include: #include <**cyg/kernel/intr.hxx**>

Returns: nothing

See Also:

Cyg_Interrupt::configure_interrupt

Name: ***Cyg_Interrupt::configure_interrupt* ( ) -** configure an interrupt

Synopsis:

```
void Cyg_Interrupt::configure_interrupt
(
  cyg_vector vector, /* interrupt vector                  */
  cyg_bool   level,  /* level or edge triggered           */
  cyg_bool   up      /* hi/lo level, rising/falling edge */
)
```

Description: This configures an interrupt for level triggering or edge, as well as hi/low and rising/fallig edge.

Include: #include <**cyg/kernel/intr.hxx**>

Returns: nothing

See Also:

Cyg_Interrupt::set_cpu

Name: ***Cyg_Interrupt::set_cpu* ( ) -** set cpu

Synopsis:

```
void Cyg_Interrupt::set_cpu
(
  cyg_vector       vector, /* interrupt vector */
  HAL_SMP_CPU_TYPE cpu     /* CPU to set       */
)
```

Description: This sets a CPU. Honestly, I have no idea what this does in an SMP system but when I find out I'll properly document it.

Include: #include <**cyg/kernel/intr.hxx**>

Returns: nothing

See Also: Cyg_Interrupt::get_cpu

Cyg_Interrupt::get_cpu

Name: ***Cyg_Interrupt::get_cpu*** **( )** - get cpu

Synopsis:

```
HAL_SMP_CPU_TYPE Cyg_Interrupt::get_cpu
(
  cyg_vector vector /* interrupt vector */
)
```

Description: This gets a CPU. Honestly, I have no idea what this does in an SMP system but when I find out I'll properly document it.

Include: #include <**cyg/kernel/intr.hxx**>

Returns: presumably the CPU attached to this interrupt.

See Also: Cyg_Interrupt::set_cpu

Cyg_Mbox::Cyg_Mbox

Name: ***Cyg_Mbox::Cyg_Mbox* ( )** - create a message box

Synopsis:

```
Cyg_Mbox::Cyg_Mbox
(
  void
)
```

Description: This creates a message box. Message boxes are of fixed depth. The depth of the message box is controlled by the compilation of the kernel. Most often the depth is 10.

Include: #include <**cyg/kernel/mbox.hxx**>

Returns: nothing

See Also: Cyg_Mbox::~Cyg_Mbox

Cyg_Mbox::~Cyg_Mbox

Name: ***Cyg_Mbox::~Cyg_Mbox* ( )** - destroy a message box

Synopsis:

```
Cyg_Mbox::~Cyg_Mbox
(
  void
)
```

Description: This destroys a message box. Be sure that no threads are using a message box before destroying it.

Include: #include <**cyg/kernel/mbox.hxx**>

Returns: nothing

See Also: Cyg_Mbox::Cyg_Mbox

Cyg_Mbox::get

Name: ***Cyg_Mbox::get* ( )** - get a message from a message box

Synopsis:

```
void * Cyg_Mbox::get
(
  void
)
```

Description: Retrieves a message from a message box. If the message box is empty this will block until a message is added.

Include: #include <**cyg/kernel/mbox.hxx**>

Returns: a pointer to the oldest message placed in the message box.

See Also: Cyg_Mbox::tryget, Cyg_Mbox::peek_item, Cyg_Mbox::put, Cyg_Mbox::tryput, Cyg_Mbox::peek

---

Cyg_Mbox::get

Name: ***Cyg_Mbox::get* ( )** - get a message from a message box with timeout

Synopsis:

```
void * Cyg_Mbox::get
(
  cyg_tick_count absolute_time /* max tick count before timeout */
)
```

Description: Retrieves a message from a message box. If the message box is empty this will block until a message is added or the system tick reaches "absolute_time".

Include: #include <**cyg/kernel/mbox.hxx**>

Returns: a pointer to the oldest message placed in the message box or a NULL pointer if the wait timed out.

See Also: Cyg_Mbox::tryget, Cyg_Mbox::peek_item, Cyg_Mbox::put, Cyg_Mbox::tryput, Cyg_Mbox::peek

---

Cyg_Mbox::tryget

Name: ***Cyg_Mbox::tryget* ( )** - get a message from a message box if one is available

Synopsis:

```
void * Cyg_Mbox::tryget
(
  void
)
```

Description: Retrieves a message from a message box is a message is available. If the message box is empty, this will return immediately with an error.

Include: #include <**cyg/kernel/mbox.hxx**>

Returns: a pointer to the oldest message placed in the message box or a NULL pointer if the message box is empty.

See Also: Cyg_Mbox::get, Cyg_Mbox::peek_item, Cyg_Mbox::put, Cyg_Mbox::tryput, Cyg_Mbox::peek

---

Cyg_Mbox::peek_item

Name: ***Cyg_Mbox::peek_item* ( )** - get a message from an mbox but don't remove from queue

Synopsis:

```
void * Cyg_Mbox::peek_item
(
  void
)
```

Description: Retrieves the address of the oldest message in a message box but does not remove the message from the message box. This will not block, if the message box is emtpy NULL is returned.

Include: #include <**cyg/kernel/mbox.hxx**>

Returns: a pointer to the oldest message placed in the message box or a NULL pointer if the message box is empty.

See Also: Cyg_Mbox::get, Cyg_Mbox::tryget, Cyg_Mbox::put, Cyg_Mbox::tryput, Cyg_Mbox::peek

---

Cyg_Mbox::put

Name: ***Cyg_Mbox::put* ( )** - place a message into a message box

Synopsis:

```
cyg_bool Cyg_Mbox::put
(
  void *item /* item to place into message box */
)
```

Description: Places a new message in a message box. If the message box is full the thread will be blocked until the the message can be placed into the message box.

Include: #include <**cyg/kernel/mbox.hxx**>

Returns: "true" if the message was placed into the message box, "false" otherwise.

See Also: Cyg_Mbox::get, Cyg_Mbox::tryget, Cyg_Mbox::peek_item, Cyg_Mbox::tryput, Cyg_Mbox::peek

---

Cyg_Mbox::put

Name: ***Cyg_Mbox::put* ( )** - place a message into a message box with a timeout

Synopsis:

```
cyg_bool Cyg_Mbox::put
(
  void           *item,        /* item to place into message box */
  cyg_tick_count absolute_time /* max tick count before timeout  */
)
```

Description: Places a new message in a message box. If the message box is full the thread will be blocked until the the message can be placed into the message box or the system time reaches "absolute_time".

Include: #include <**cyg/kernel/mbox.hxx**>

Returns: "true" if the message was placed into the message box, "false" otherwise.

See Also: Cyg_Mbox::get, Cyg_Mbox::tryget, Cyg_Mbox::peek_item, Cyg_Mbox::tryput, Cyg_Mbox::peek

---

Cyg_Mbox::tryput

Name: ***Cyg_Mbox::tryput* ( )** - place a message into a message box if space is available

Synopsis:

```
cyg_bool Cyg_Mbox::tryput
(
  void *item /* item to place into message box */
)
```

Description: Places a new message in a message box. If the message box is full the message will not be placed in the message box and the function will return with an error status.

Include: #include <**cyg/kernel/mbox.hxx**>

Returns: "true" if the message was placed into the message box, "false" otherwise.

See Also: Cyg_Mbox::get, Cyg_Mbox::tryget, Cyg_Mbox::peek_item, Cyg_Mbox::put, Cyg_Mbox::peek

---

Cyg_Mbox::peek

Name: ***Cyg_Mbox::peek* ( )** - gets the number of messages currently in the message queue

Synopsis:

```
cyg_count32 Cyg_Mbox::peek
(
  void
)
```

Description: This reports the number of messages currently in the message queue waiting to be processed.

Include: #include <**cyg/kernel/mbox.hxx**>

Returns: the number of messages currently in the message queue.

See Also: Cyg_Mbox::get, Cyg_Mbox::tryget, Cyg_Mbox::peek_item, Cyg_Mbox::put, Cyg_Mbox::tryput

---

Cyg_Mbox::waiting_to_get

Name: ***Cyg_Mbox::waiting_to_get* ( )** - reports if any threads are waiting to get a message from this mbox

Synopsis:

```
cyg_bool Cyg_Mbox::waiting_to_get
(
  void
)
```

Description: This reports whether any threads are being blocked waiting to get a message from this message box.

Include: #include <**cyg/kernel/mbox.hxx**>

Returns: "true" if any threads are waiting for a message, "false" is no threads are waiting for a message from the queue.

See Also: Cyg_Mbox::waiting_to_put

Cyg_Mbox::waiting_to_put

Name: ***Cyg_Mbox::waiting_to_put*** **( )** - reports if any threads are waiting to place a message into this mbox

Synopsis:

```
cyg_bool Cyg_Mbox::waiting_to_put
(
  void
)
```

Description: This reports whether any threads are being blocked waiting to place a message into this message box.

Include: #include <**cyg/kernel/mbox.hxx**>

Returns: "true" if any threads are waiting to place a message into this message box, "false" is no threads are waiting to place a message into this message box.

See Also: Cyg_Mbox::waiting_to_get

Cyg_Mempool_Fixed::Cyg_Mempool_Fixed

Name: *Cyg_Mempool_Fixed::Cyg_Mempool_Fixed* **( )** - create fixed block memory heap

Synopsis:

```
Cyg_Mempool_Fixed::Cyg_Mempool_Fixed
(
  cyg_uint8    *base,     /* base address of heap        */
  cyg_int32    size,      /* size of heap                */
  CYG_ADDRWORD alloc_unit /* fixed allocation block size */
)
```

Description: This creates a heap used to allocate fixed blocks. Unlike the traditional malloc() and free() functions this provides access to fixed blocks of memory. The advantage to this is that there is no fragmentation with fixed blocks of memory, and allocation is faster since linked lists don't have to be searched.

Include: #include <**cyg/memalloc/memfixed.hxx**>

Returns: nothing

See Also: Cyg_Mempool_Fixed::~Cyg_Mempool_Fixed

Cyg_Mempool_Fixed::~Cyg_Mempool_Fixed

Name: *Cyg_Mempool_Fixed::~Cyg_Mempool_Fixed* **( )** - destroy fixed size memory pool

Synopsis:

```
Cyg_Mempool_Fixed::~Cyg_Mempool_Fixed
(
  void
)
```

Description: This destroys a fixed block memory pool. Be certain that there are no allocated blocks before destroying a heap.

Include: #include <**cyg/memalloc/memfixed.hxx**>

Returns: nothing

See Also: Cyg_Mempool_Fixed::Cyg_Mempool_Fixed, Cyg_Mempool_Fixed::get_status

Cyg_Mempool_Fixed::try_alloc

Name: *Cyg_Mempool_Fixed::try_alloc* **( )** - allocate a fixed block of memory, nonblocking

Synopsis:
```
cyg_uint8 * Cyg_Mempool_Fixed::try_alloc
(
  void
)
```

Description: This allocates a block of memory from a fixed sized heap. If there is no memory available in the heap, this will return immediately with a failure.

Include: #include <**cyg/memalloc/memfixed.hxx**>

Returns: a newly allocated fixed size block of memory or NULL if there was no memory available.

See Also: Cyg_Mempool_Fixed::Cyg_Mempool_Fixed, Cyg_Mempool_Fixed::alloc, Cyg_Mempool_Fixed::free, Cyg_Mempool_Fixed::get_status

Cyg_Mempool_Fixed::alloc

Name: *Cyg_Mempool_Fixed::alloc* **( )** - allocate a fixed block size, block thread if necessary

Synopsis:
```
cyg_uint8 * Cyg_Mempool_Fixed::alloc
(
  void
)
```

Description: This allocates a fixed size block of memory. The size of the memory block allocated is defined by the heap. If there isn't a block of memory available this will block the thread until there is one made available.

Include: #include <**cyg/memalloc/memfixed.hxx**>

Returns: a pointer to the newly allocated block.

See Also: Cyg_Mempool_Fixed::Cyg_Mempool_Fixed, Cyg_Mempool_Fixed::try_alloc, Cyg_Mempool_Fixed::free, Cyg_Mempool_Fixed::get_status

Cyg_Mempool_Fixed::alloc

Name: *Cyg_Mempool_Fixed::alloc* **( )** - allocate a fixed block size with timeout

Synopsis:

```
cyg_uint8 * Cyg_Mempool_Fixed::alloc
(
  cyg_tick_count absolute_time /* absolute delay timeout */
)
```

Description: This allocates a fixed block of memory from a fixed size heap. If there is not a block available the thread will be blocked until "absolute_time" or until there is a block of memory available.

Include: #include <**cyg/memalloc/memfixed.hxx**>

Returns: a newly allocated fixed size block of memory or NULL if there were no memory made available within the timeout period.

See Also: Cyg_Mempool_Fixed::Cyg_Mempool_Fixed, Cyg_Mempool_Fixed::try_alloc, Cyg_Mempool_Fixed::free, Cyg_Mempool_Fixed::get_status

---

Cyg_Mempool_Fixed::free

Name: ***Cyg_Mempool_Fixed::free* ( )** - return a block of memory to a fixed sized heap

Synopsis:

```
cyg_bool Cyg_Mempool_Fixed::free
(
  cyg_uint8 *pointer /* pointer to a fixed sized block */
)
```

Description: Returns a block of memory to a fixed size heap that was previously allocated from that fixed sized heap.

Include: #include <**cyg/memalloc/memfixed.hxx**>

Returns: nothing

See Also: Cyg_Mempool_Fixed::Cyg_Mempool_Fixed, Cyg_Mempool_Fixed::alloc, Cyg_Mempool_Fixed::try_alloc, Cyg_Mempool_Fixed::get_status

---

Cyg_Mempool_Fixed::get_status

Name: ***Cyg_Mempool_Fixed::get_status* ( )** - get status on a heap

Synopsis:

```
void Cyg_Mempool_Fixed::get_status
(
  cyg_mempool_status_flag_t flags,   /* flags  */
  Cyg_Mempool_Status        &status, /* status */
)
```

Description: This returns information about a memory pool. Which elements that are returned are dependant on what flags are set. The flags are shown below. See the implementation of Cyg_Mempool_Status to see what this is all about.

CYG_MEMPOOL_STAT_ARENABASE - base address of entire pool

CYG_MEMPOOL_STAT_ARENASIZE - total size of entire pool

CYG_MEMPOOL_STAT_FREEBLOCKS - number of blocks free to use

CYG_MEMPOOL_STAT_TOTALALLOCATED - total allocated space in bytes

CYG_MEMPOOL_STAT_TOTALFREE - total number of bytes unusued

CYG_MEMPOOL_STAT_BLOCKSIZE - block size of fixed block

CYG_MEMPOOL_STAT_MAXFREE - size of largest unused block

CYG_MEMPOOL_STAT_WAITING - any threads waiting?

CYG_MEMPOOL_STAT_ORIGBASE - original base of pool

CYG_MEMPOOL_STAT_ORIGSIZE - original size of pool

CYG_MEMPOOL_STAT_MAXOVERHEAD - maximum overhead used by the allocator

Include: #include <**cyg/memalloc/memfixed.hxx**>
#include <**cyg/memalloc/common.hxx**>

Returns: nothing

See Also: Cyg_Mempool_Fixed::Cyg_Mempool_Fixed

Cyg_Mempool_Variable::Cyg_Mempool_Variable

Name: ***Cyg_Mempool_Variable::Cyg_Mempool_Variable*** **( )** - create a variable heap

Synopsis:

```
Cyg_Mempool_Variable::Cyg_Mempool_Variable
(
  cyg_uint8 *base,    /* base adress of heap */
  cyg_int32 size,     /* size of heap        */
  cyg_int32 alignment /* alignment           */
)
```

Description: This creates a heap of memory for dynamic memory allocation. This provides equivalents to malloc() and free().

Include: #include <**cyg/memalloc/memvar.hxx**>

Returns: nothing

See Also: Cyg_Mempool_Variable::~Cyg_Mempool_Variable, Cyg_Mempool_Variable::get_status

Cyg_Mempool_Variable::~Cyg_Mempool_Variable

Name: ***Cyg_Mempool_Variable::~Cyg_Mempool_Variable*** **( )** - destroy variable heap

Synopsis:

```
Cyg_Mempool_Variable::~Cyg_Mempool_Variable
(
  void
)
```

Description: This destroys a heap of memory. Before destroying a mempool, be sure no memory is currently allocated from that heap.

Include: #include <**cyg/memalloc/memvar.hxx**>

Returns: nothing

See Also: Cyg_Mempool_Variable::Cyg_Mempool_Variable

Cyg_Mempool_Variable::try_alloc

Name: ***Cyg_Mempool_Variable::try_alloc*** **( )** - allocate a block of memory

Synopsis:

```
cyg_uint8 * Cyg_Mempool_Variable::try_alloc
(
  cyg_int32 size /* number of bytes to allocate */
)
```

Description: This allocates an arbitrary block of memory from a heap. This will not block the calling thread under any circumstances.

Include: #include <**cyg/memalloc/memvar.hxx**>

Returns: a pointer to the allocated memory or NULL if the memory could not be allocated.

See Also: Cyg_Mempool_Variable::alloc, Cyg_Mempool_Variable::resize_alloc, Cyg_Mempool_Variable::free

---

Cyg_Mempool_Variable::alloc

Name: ***Cyg_Mempool_Variable::alloc* ( )** - allocate a block of memory, block until memory available

Synopsis:

```
cyg_uint8 * Cyg_Mempool_Variable::alloc
(
  cyg_int32 size /* bytes to allocate */
)
```

Description: This allocates an arbitrary block of memory from a heap. If the request cannot be satisified, this will block the thread until enough memory can be allocated to satisfy the request.

Include: #include <**cyg/memalloc/memvar.hxx**>

Returns: a pointer to the allocated memory or NULL if the memory could not be allocated.

See Also: Cyg_Mempool_Variable::try_alloc, Cyg_Mempool_Variable::resize_alloc, Cyg_Mempool_Variable::free

---

Cyg_Mempool_Variable::alloc

Name: ***Cyg_Mempool_Variable::alloc* ( )** - allocate a block of memory with timeout

Synopsis:

```
cyg_uint8 * Cyg_Mempool_Variable::alloc
(
  cyg_int32      size,          /* bytes to allocate      */
  cyg_tick_count absolute_time /* absolute delay timeout */
)
```

Description: This allocates an arbitrary block of memory from the heap. If there is not enough memory available to satisfy the request, the thread will be blocked until "absolute_time" or until there is sufficient memory available to satisfy the request.

Include: #include <**cyg/memalloc/memvar.hxx**>

Returns: a pointer to the allocated memory or NULL if the memory could not be allocated.

See Also: Cyg_Mempool_Variable::try_alloc, Cyg_Mempool_Variable::resize_alloc, Cyg_Mempool_Variable::free

---

Cyg_Mempool_Variable::resize_alloc

Name: ***Cyg_Mempool_Variable::resize_alloc*** **( )** - resize a previously allocated block of memory

Synopsis:

```
cyg_uint8 * Cyg_Mempool_Variable::resize_alloc
(
  cyg_uint8 *alloc_ptr,   /* previously allocated ptr */
  cyg_int32 newsize,      /* new desired size         */
  cyg_int32 *oldsize=NULL /* receives old size        */
)
```

Description: Note that this is **not** the same as the standard C realloc() function. The behaviour of this function is undefined if "alloc_ptr" is set to NULL "newsize" is set to 0.

Attempts to resize a previous allocation. It the previous allocation cannot be resized, this will fail. It will not attempt to allocate new memory if the previous allocation fails.

Include: #include <**cyg/memalloc/memvar.hxx**>

Returns: the original alloc_ptr if successful, NULL on failure.

See Also: Cyg_Mempool_Variable::try_alloc, Cyg_Mempool_Variable::alloc, Cyg_Mempool_Variable::free

---

Cyg_Mempool_Variable::free

Name: ***Cyg_Mempool_Variable::free*** **( )** - free an allocated block of memory

Synopsis:

```
cyg_bool Cyg_Mempool_Variable::free
(
  cyg_uint8 *ptr,  /* ptr to free */
  cyg_int32 size=0 /* size        */
)
```

Description: This frees a previously allocated block of memory. The "size" parameter doesn't appear to be actually used.

Include: #include <**cyg/memalloc/memvar.hxx**>

Returns:

See Also: Cyg_Mempool_Variable::try_alloc, Cyg_Mempool_Variable::alloc, Cyg_Mempool_Variable::resize_alloc

---

Cyg_Mempool_Variable::get_status

Name: *Cyg_Mempool_Variable::get_status* **( )** - get status on a heap

Synopsis:

```
void Cyg_Mempool_Variable::get_status
(
  cyg_mempool_status_flag_t flags,   /* flags  */
  Cyg_Mempool_Status        &status, /* status */
)
```

Description: This returns information about a memory pool. Which elements that are returned are dependant on what flags are set. The flags are shown below. See the implementation of Cyg_Mempool_Status to see what this is all about.

CYG_MEMPOOL_STAT_ARENABASE - base address of entire pool

CYG_MEMPOOL_STAT_ARENASIZE - total size of entire pool

CYG_MEMPOOL_STAT_FREEBLOCKS - number of blocks free to use

CYG_MEMPOOL_STAT_TOTALALLOCATED - total allocated space in bytes

CYG_MEMPOOL_STAT_TOTALFREE - total number of bytes unusued

CYG_MEMPOOL_STAT_BLOCKSIZE - block size of fixed block

CYG_MEMPOOL_STAT_MAXFREE - size of largest unused block

CYG_MEMPOOL_STAT_WAITING - any threads waiting?

CYG_MEMPOOL_STAT_ORIGBASE - original base of pool

CYG_MEMPOOL_STAT_ORIGSIZE - original size of pool

CYG_MEMPOOL_STAT_MAXOVERHEAD - maximum overhead used by the allocator

Include: #include <**cyg/memalloc/memvar.hxx**>
#include <**cyg/memalloc/common.hxx**>

Returns: nothing

See Also: Cyg_Mempool_Variable::Cyg_Mempool_Variable

Cyg_Mutex::Cyg_Mutex

Name: ***Cyg_Mutex::Cyg_Mutex*** **( )** - create a mutex

Synopsis:

```
Cyg_Mutex::Cyg_Mutex
(
  void
)
```

Description: Creates a mutex. The mutex is created in an unlocked state. If the kernel was compiled without priority inversion the effective protocol is Cyg_Mutex::NONE.

If the kernel was compiled with priority inversion enabled, the protocol of the mutex depends on how the kernel was compiled. More data will be forthcoming on this in later revisions of the documentation. It is probably best to use the other Cyg_Mutex constructor explicitly if priority inversion is enabled.

Include: #include <**cyg/kernel/mutex.hxx**>

Returns: nothing

See Also: Cyg_Mutex::~Cyg_Mutex

Cyg_Mutex::Cyg_Mutex

Name: ***Cyg_Mutex::Cyg_Mutex*** **( )** - create a mutex with a specified protocol

Synopsis:

```
Cyg_Mutex::Cyg_Mutex
(
  cyg_protcol protocol /* priority inheritence protocol */
)
```

Description: Creates a mutex with a protocol. Cyg_Mutex::NONE, Cyg_Mutex::INHERIT, or Cyg_Mutex::CEILING are the possible values. This is only valid if the kernel is compiled to support priority inversion.

Include: #include <**cyg/kernel/mutex.hxx**>

Returns: nothing

See Also: Cyg_Mutex::~Cyg_Mutex, Cyg_Mutex::set_protocol

Cyg_Mutex::~Cyg_Mutex

Name: ***Cyg_Mutex::~Cyg_Mutex* ( )** - destroy a mutex

Synopsis:

```
Cyg_Mutex::~Cyg_Mutex
(
  void
)
```

Description: Destroys a mutex. If other threads are waiting for this mutex they will not be released, so be sure that no threads are waiting on this mutex before destroying it.

Include: #include <**cyg/kernel/mutex.hxx**>

Returns: nothing

See Also: Cyg_Mutex::Cyg_Mutex

Cyg_Mutex::lock

Name: ***Cyg_Mutex::lock* ( )** - lock a mutex or wait until it can be locked

Synopsis:

```
cyg_bool Cyg_Mutex::lock
(
  void
)
```

Description: Locks a mutex. If the mutex is not available this will wait until the mutex is free. Note that only the owner of a locked mutex can release it.

Include: #include <**cyg/kernel/mutex.hxx**>

Returns: "true" if the mutex was locked by the calling thread "false" if the mutex could not be locked.

See Also: Cyg_Mutex::trylock, Cyg_Mutex::unlock, Cyg_Mutex::release, Cyg_Mutex::get_owner

---

Cyg_Mutex::trylock

Name: ***Cyg_Mutex::trylock* ( )** - lock a mutex if it's free

Synopsis:	`cyg_bool Cyg_Mutex::trylock ( void )`
Description:	Locks a mutex only if it's free. If the mutex is not available this call will return immediately.
Include:	#include <**cyg/kernel/mutex.hxx**>
Returns:	"true" if the mutex was locked by the calling thread "false" if the mutex could not be locked or has been allocated by another thread.
See Also:	Cyg_Mutex::lock, Cyg_Mutex::unlock, Cyg_Mutex::release, Cyg_Mutex::get_owner

---

Cyg_Mutex::unlock

Name:	***Cyg_Mutex::unlock* ( )** - unlock a mutex
Synopsis:	`void Cyg_Mutex::unlock ( void )`
Description:	Unlocks a mutex that was previously allocated by the calling thread. If the calling thread is not the owner, the behavior is undefined.
Include:	#include <**cyg/kernel/mutex.hxx**>
Returns:	nothing
See Also:	Cyg_Mutex::trylock, Cyg_Mutex::lock, Cyg_Mutex::release

---

Cyg_Mutex::release

Name:	***Cyg_Mutex::release* ( )** - release all threads waiting on a mutex
Synopsis:	`void Cyg_Mutex::release ( void )`
Description:	Releases all threads waiting on a mutex. Any thread that is waiting on the mutex will not receive ownership of the mutex but will return an error if they are waiting on the mutex.

Include:	#include <**cyg/kernel/mutex.hxx**>
Returns:	nothing
See Also:	Cyg_Mutex::trylock, Cyg_Mutex::lock, Cyg_Mutex::unlock

---

Cyg_Mutex::set_ceiling

Name: ***Cyg_Mutex::set_ceiling* ( )** - set the max priority to be inherited

Synopsis:

```
void Cyg_Mutex::set_ceiling
(
  cyg_priority priority /* ceiling priority */
)
```

Description: Set the priority ceiling of the mutex. If this mutex has the protocol of Cyg_Mutex::CEILING any thread that owns this mutex will have it's priority temporarily set to the value specified here.

It's a good idea to use this function if you use priority inheritence. The kernel will support default values, and it's typically 0, but it's good programming practice to explicity set it in your application software.

Include: #include <**cyg/kernel/mutex.hxx**>

Returns: nothing

See Also: Cyg_Mutex::get_ceiling

Cyg_Mutex::get_ceiling

Name: ***Cyg_Mutex::get_ceiling* ( )** - get the priority ceiling of this mutex

Synopsis:

```
cyg_priority Cyg_Mutex::get_ceiling
(
  void
)
```

Description: This gets the priority ceiling of the mutex. This is only meaningful if the protocol of the mutex is of type Cyg_Mutex::CEILING.

Include: #include <**cyg/kernel/mutex.hxx**>

Returns: the priority ceiling of the mutex.

See Also: Cyg_Mutex::set_ceiling

Cyg_Mutex::get_owner

Name: ***Cyg_Mutex::get_owner*** **( )** - get the current owner of a mutex

Synopsis:

```
Cyg_Thread *Cyg_Mutex::get_owner
(
  void
)
```

Description: Gets the current owner of the mutex.

Include: #include <**cyg/kernel/mutex.hxx**>

Returns: a Cyg_Thread pointer to the current owner of the mutex or NULL if the mutex is not owned by anybody.

See Also: Cyg_Mutex::lock, Cyg_Mutex::trylock

Cyg_Mutex::set_protocol

Name: ***Cyg_Mutex::set_protocol*** **( )** - set the protocol of a mutex

Synopsis:

```
void Cyg_Mutex::set_protocol
(
  cyg_protcol new_protocol
)
```

Description: This sets the protocol of a mutex. Valid values are Cyg_Mutex::NONE, Cyg_Mutex::INHERIT or Cyg_Mutex::CEILING.

Cyg_Mutex::NONE, no priority inheritance

Cyg_Mutex::INHERIT, inherit priority of thread currently holding mutex

Cyg_Mutex::CEILING, inherit ceiling priority of mutex

Include: #include <**cyg/kernel/mutex.hxx**>

Returns: nothing

See Also: Cyg_Mutex::Cyg_Mutex, Cyg_Mutex::set_ceiling, Cyg_Mutex::get_ceiling

Cyg_Scheduler::get_sched_lock

Name: ***Cyg_Scheduler::get_sched_lock* ( )** - get the lock count of a thread

Synopsis:

```
static cyg_ucount32 Cyg_Scheduler::get_sched_lock
(
  void
)
```

Description: Gets the lock count on the current thread. Will return 0 if this thead is not locked. Note that this class function is static.

Include: #include <**cyg/kernel/sched.hxx**>
#include <**cyg/kernel/sched.inl**>

Returns: The number of times the calling thread has been locked.

See Also: Cyg_Scheduler::lock, Cyg_Scheduler::unlock

Cyg_Scheduler::lock

Name: ***Cyg_Scheduler::lock* ( )** - lock a thread

Synopsis:

```
static void Cyg_Scheduler::lock
(
  void
)
```

Description: Prevents the calling thread from being preempted by another thread. Note that this class function is static. A thread can be locked multiple times. Each call to Cyg_Scheduler::lock() must be paired with a call to Cyg_Scheduler::unlock() to unlock the scheduler.

Include: #include <**cyg/kernel/sched.hxx**>
#include <**cyg/kernel/sched.inl**>

Returns: nothing

See Also: Cyg_Scheduler::get_sched_lock, Cyg_Scheduler::unlock

Cyg_Scheduler::unlock

Name: ***Cyg_Scheduler::unlock*** **( )** - unlock a thread

Synopsis:

```
static void Cyg_Scheduler::unlock
(
  void
)
```

Description: This unlocks a thread so that it can be preempted by another thread. Note that for each call to Cyg_Scheduler::lock there has to be a call to this class function to actually unlock the thread.

Include: #include <**cyg/kernel/sched.hxx**>
#include <**cyg/kernel/sched.inl**>

Returns: nothing

See Also: Cyg_Scheduler::get_sched_lock, Cyg_Scheduler::lock

Cyg_Thread::Cyg_Thread

Name: *Cyg_Thread::Cyg_Thread* **( )** - contructor to create a new thread

Synopsis:

```
Cyg_Thread::Cyg_Thread
(
  CYG_ADDRWORD     sched_info,     /* Scheduling parameter(s)     */
  cyg_thread_entry *entry,         /* entry point function        */
  CYG_ADDRWORD     entry_data,     /* entry data                  */
  char             *name,          /* thread name                 */
  CYG_ADDRESS      stack_base = 0, /* stack base, NULL = allocate */
  cyg_ucount32     stack_size = 0  /* stack size, 0 = use default */
)
```

Description: This is the constructor to create a new thread. The "sched_info" parameter is usually just the priority, although this may change depending on the scheduler that is being used. The "name" parameter is optional but it's strongly recommended that it be used. If you do use the "name" parameter it MUST be a constant string since the string will not be copied, only the pointer to it will be.

NOTE: stack_base and stack_size CANNOT be 0 in this current implementation although this may change in future versions.

The entry function is of type: void cyg_thread_entry(CYG_ADDRWORD data).

Include: #include <**cyg/kernel/thread.hxx**>

Returns: nothing

See Also: Cyg_Thread::exit

Cyg_Thread::exit

Name: *Cyg_Thread::exit* **( )** - terminate calling thread

Synopsis:

```
static void Cyg_Thread::exit
(
  void
)
```

Description: This terminates the calling thread. Before exiting a thread, be sure to free any resources you may have allocated such as mutexes, semaphores, memory, etc. This will also call the destructors if there are any before exiting.

Include: #include <**cyg/kernel/thread.hxx**>

Returns: nothing - this function will not return

See Also:

Cyg_Thread::suspend

Name: ***Cyg_Thread::suspend*** **( )** - suspend a thread

Synopsis:

```
void Cyg_Thread::suspend
(
  void
)
```

Description: This suspends a thread. For every call to suspend a thread, there must be a matching call to resume it with Cyg_Thread::resume(). Suspending a thread may prevent resources that the thread has from being released, so care must be taking in using this function.

Include: #include <**cyg/kernel/thread.hxx**>

Returns: nothing

See Also: Cyg_Thread::resume, Cyg_Thread::force_resume

Cyg_Thread::resume

Name: ***Cyg_Thread::resume*** **( )** - resume a suspended thread

Synopsis:

```
void Cyg_Thread::resume
(
  void
)
```

Description: This resumes a thread that has been suspended. If the thread has been suspended multiple times, the thread will have to be resumed the same number of times before it can run again.

Include: #include <**cyg/kernel/thread.hxx**>

Returns: nothing

See Also: Cyg_Thread::suspend, Cyg_Thread::force_resume

Cyg_Thread::force_resume

Name: ***Cyg_Thread::force_resume*** **( )** - force a suspended thread to be resumed

Synopsis:

```
void Cyg_Thread::force_resume
(
  void
)
```

Description: This resumes a thread that has been suspended regardless of how many times it's been suspended.

Include: #include <**cyg/kernel/thread.hxx**>

Returns: nothing

See Also: Cyg_Thread::suspend, Cyg_Thread::resume

Cyg_Thread::kill

Name: *Cyg_Thread::kill* **( )** - kill a thread

Synopsis:

```
void Cyg_Thread::kill
(
  void
)
```

Description: This kills the calling thread. Before killing a thread, be sure to free any resources you may have allocated such as mutexes, semaphores, memory, etc.

Note that this is probably a bad way of shutting down a thread. If possible, send a signal to the thread to have it shut itself down so it can free any resources it's allocated.

Include: #include <**cyg/kernel/thread.hxx**>

Returns: nothing

See Also:

Cyg_Thread::release

Name: *Cyg_Thread::release* **( )** - force a thread to wake up with the reason of BREAK

Synopsis:

```
void Cyg_Thread::release
(
  void
)
```

Description: This wakes a thread up from a DELAY. It may also wake the thread up on a blocking wait for a semaphore, mutex, etc. It is the responsibility of the thread being woken to detect that it's been awoken

Include: #include <**cyg/kernel/thread.hxx**>

Returns: nothing

See Also:

Cyg_Thread::yield

Name: *Cyg_Thread::yield* **( )** - yield the cpu to another thread

Synopsis:

```
static void Cyg_Thread::yield
(
  void
)
```

Description: This yields the current thread to another, usually of the same priority - although this depends on the scheduler implementation.

Include: #include <**cyg/kernel/thread.hxx**>
#include <**cyg/kernel/thread.inl**>

Returns: nothing

See Also:

Cyg_Thread::self

Name: *Cyg_Thread::self* **( )** - get the "this" pointer of the calling thread

Synopsis:

```
static Cyg_Thread *self
(
  void
)
```

Description: This function returns the thread's "this" pointer. This is most useful when in a C context and doesn't serve much purpose when working with C++ other than defensive programming.

You can use this function to find out which thread was interrupted in an interrupt context.

Include: #include <**cyg/kernel/thread.hxx**>
#include <**cyg/kernel/thread.inl**>

Returns: Pointer to the thread's "this" pointer.

See Also:

Cyg_Thread::set_priority

Name: *Cyg_Thread::set_priority* **( )** - set priority of a thread

Synopsis:

```
void Cyg_Thread::set_priority
(
  cyg_priority new_priority  /* new priority */
)
```

Description: Changes the priority of a given thread. Note there is no error checking, so be careful to check ranges when using this to change the priority of a thread.

Include: #include <**cyg/kernel/thread.hxx**>

Returns: nothing

See Also: Cyg_Thread::get_priority, Cyg_Thread::get_current_priority

---

Cyg_Thread::get_priority

Name: *Cyg_Thread::get_priority* **( )** - get the set priority of a thread

Synopsis:

```
cyg_priority get_priority
(
  void
)
```

Description: This returns the set priority of a given thread.

Include: #include <**cyg/kernel/thread.hxx**>
#include <**cyg/kernel/thread.inl**>

Returns: The set priority of a given thread.

See Also: Cyg_Thread::set_priority, Cyg_Thread::get_current_priority

---

Cyg_Thread::get_current_priority

Name: ***Cyg_Thread::get_current_priority* ( )** - get the current priority of a thread

Synopsis:

```
cyg_priority Cyg_Thread::get_current_priority
(
  void
)
```

Description: This returns the current priority of a given thread. This is normally what Cyg_Thread::get_priority returns but if the thread has inherited another priority, the inherited priority will be returned instead.

Include: #include <**cyg/kernel/thread.hxx**>
#include <**cyg/kernel/thread.inl**>

Returns: the current priority of a given thread.

See Also: Cyg_Thread::set_priority, Cyg_Thread::get_priority

Cyg_Thread::delay

Name: ***Cyg_Thread::delay* ( )** - delay a thread

Synopsis:

```
void Cyg_Thread::delay
(
  cyg_tick_count delay /* number of ticks to delay */
)
```

Description: This delays a thread for a specified number of ticks.

Include: #include <**cyg/kernel/thread.hxx**>

Returns: nothing

See Also:

Cyg_HardwareThread::get_stack_base

Name: ***Cyg_HardwareThread::get_stack_base* ( )** - get base address of a thread's stack

Synopsis:

```
CYG_ADDRESS Cyg_HardwareThread::get_stack_base
(
  void
)
```

Description: This returns the base address of a stack of the given thread.

Include: #include <**cyg/kernel/thread.hxx**>
#include <**cyg/kernel/thread.inl**>

Returns: the base address of the given thread's stack

See Also: Cyg_HardwareThread::get_stack_size

Cyg_HardwareThread::get_stack_size

Name: *Cyg_HardwareThread::get_stack_size* **( )** - get the size of a thread's stack

Synopsis:

```
cyg_uint32 Cyg_HardwareThread::get_stack_size
(
  void
)
```

Description: This returns the size of a given thread's stack in bytes.

Include: #include <**cyg/kernel/thread.hxx**>
#include <**cyg/kernel/thread.inl**>

Returns: the size in bytes of the given thread's stack

See Also: Cyg_HardwareThread::get_stack_base

Cyg_HardwareThread::measure_stack_usage

Name: *Cyg_HardwareThread::measure_stack_usage* **( )** - measure a stack's usage

Synopsis:

```
cyg_uint32 Cyg_HardwareThread::measure_stack_usage
(
  void
)
```

Description: This function measures the number of bytes that have been used for a given thread. This can help you tune your stack sizes so there is less memory usage although you should never tune your stacks so that this function returns 0.

Note that this function only returns how much stack has been consumed for a given thread at the current time the function was invoked. There is no guarentee that more stack will not be consumed later.

Include: #include <**cyg/kernel/thread.hxx**>
#include <**cyg/kernel/thread.inl**>

Returns: the number of bytes that have been used by the stack of a given thread.

See Also: Cyg_HardwareThread::get_stack_base, Cyg_HardwareThread::get_stack_size

---

Cyg_Thread::new_data_index

Name: *Cyg_Thread::new_data_index* **( )** - gets a new data index for per thread data

Synopsis:

```
static cyg_data_index Cyg_Thread::new_data_index
(
  void
)
```

Description: Gets a new index for per thread data. The index is globally allocated for each thread.

Include: #include <**cyg/kernel/thread.hxx**>

Returns: a new (free) index that can be used to store a word of data

See Also: Cyg_Thread::free_data_index, Cyg_Thread::get_data, Cyg_Thread::get_data_ptr, Cyg_Thread::set_data

---

Cyg_Thread::free_data_index

Name: ***Cyg_Thread::free_data_index* ( )** - free a data index for per thread data

Synopsis:

```
static void Cyg_Thread::free_data_index
(
  cyg_data_index index /* index to free */
)
```

Description: This globally frees an index that was used for per thread data.

Include: #include <**cyg/kernel/thread.hxx**>

Returns: nothing

See Also: Cyg_Thread::new_data_index, Cyg_Thread::get_data, Cyg_Thread::get_data_ptr, Cyg_Thread::set_data

---

Cyg_Thread::get_data

Name: ***Cyg_Thread::get_data* ( )** - get per thread data

Synopsis:

```
static CYG_ADDRWORD Cyg_Thread::get_data
(
  cyg_data_index index /* data index */
)
```

Description: This function reads per thread data.

Include: #include <**cyg/kernel/thread.hxx**>
#include <**cyg/kernel/thread.inl**>

Returns: per thread data stored at the given index.

See Also: Cyg_Thread::get_data_ptr, Cyg_Thread::set_data

Cyg_Thread::get_data_ptr

Name: ***Cyg_Thread::get_data_ptr* ( )** - get per thread data pointer

Synopsis:

```
static CYG_ADDRWORD *Cyg_Thread::get_data_ptr
(
  Cyg_Thread::cyg_data_index index /* data index */
)
```

Description: Gets the pointer to per thread data of the calling thread. This can be used to read or write the per thread data.

Include: #include <**cyg/kernel/thread.hxx**>
#include <**cyg/kernel/thread.inl**>

Returns: a pointer to the per thread data.

See Also: Cyg_Thread::get_data, Cyg_Thread::set_data

Cyg_Thread::set_data

Name: ***Cyg_Thread::set_data*** **( )** - set per thread data

Synopsis:

```
void Cyg_Thread::set_data
(
  cyg_data_index index, /* index of the per thread data */
  CYG_ADDRWORD   data   /* data to write                */
);
```

Description: This function writes per thread data.

Include: #include <**cyg/kernel/thread.hxx**>
#include <**cyg/kernel/thread.inl**>

Returns: nothing

See Also: Cyg_Thread::get_data, Cyg_Thread::get_data_ptr

Cyg_Thread::add_destructor

Name: ***Cyg_Thread::add_destructor*** **( )** - add a thread destructor

Synopsis:

```
cyg_bool Cyg_Thread::add_destructor
(
  destructor_fn fn,  /* call back destructor function */
  CYG_ADDRWORD  data /* data to pass to destructor    */
)
```

Description: This adds a destructor to the thread. All the destructors will be called by Cyg_Thread::exit (). The destructor callback has the following prototype void destructor_fn (CYG_ADDRWORD).

Include: #include <**cyg/kernel/thread.hxx**>
#include <**cyg/kernel/thread.inl**>

Returns: "true" if the destructor was added, "false" if the destructor could not be added.

See Also: Cyg_Thread::rem_destructor, Cyg_Thread::exit

Cyg_Thread::rem_destructor

Name: ***Cyg_Thread::rem_destructor*** **( )** - remove a thread destructor

Synopsis:

```
cyg_bool Cyg_Thread::rem_destructor
(
  destructor_fn fn,   /* destructor to remove                     */
  CYG_ADDRWORD  data /* data that was to be passed to the destructor */
)
```

Description: Removes a destructor from a thread. The destructor callback and the data must be match those of the constructor otherwise the destructor will not be deleted.

Include: #include <**cyg/kernel/thread.hxx**>
#include <**cyg/kernel/thread.inl**>

Returns: "true" if the destructor was removed, "false" if the destructor could not be removed.

See Also: Cyg_Thread::add_destructor

Cyg_Thread::register_exception

Name: ***Cyg_Thread::register_exception* ( )** - register an exception handler

Synopsis:

```
static void Cyg_Thread::register_exception
(
  cyg_code                exception_number, /* exception number */
  cyg_exception_handler handler,            /* handler function */
  CYG_ADDRWORD            data,             /* data argument    */
  cyg_exception_handler **old_handler,      /* handler function */
  CYG_ADDRWORD            *old_data         /* data argument    */
)
```

Description: This registers an exception handler for the calling thread.

Include: #include <**cyg/kernel/thread.hxx**>
#include <**cyg/kernel/thread.inl**>

Returns: nothing

See Also: Cyg_Thread::deregister_exception

Cyg_Thread::deregister_exception

Name: ***Cyg_Thread::deregister_exception* ( )** - deregister an exception

Synopsis:

```
static void deregister_exception
(
  cyg_code exception_number /* exception number */
)
```

Description: This deregisters an exception handler for the calling thread.

Include: #include <**cyg/kernel/thread.hxx**>
#include <**cyg/kernel/thread.inl**>

Returns: nothing

See Also: Cyg_Thread::register_exception