11.2. TIMER_WAKE_WHEN

Wake at a specified wall time.

Synopsis

```
timer_wake_when( clock )
```

Input Parameters

```
clock       : clock_buf    time and date to wake
```

Output Parameters

```
<none>
```

Completion Status

OK	timer_wake_when operation successful
ILLEGAL_USE	operation not callable from XSR or ISR
INVALID_PARAMETER	a parameter refers to an illegal address
INVALID_CLOCK	invalid clock value

Description

This operation causes the calling task to be blocked up until a given date and time. The task is woken at this time, and is returned a successful completion status. The kernel checks the supplied clock_buf data for validity. The exact structure of that data is language binding dependent.

If the node clock is set while the timer is running, the wall time at which the task is woken remains valid. If the node time is set to after the timer wake time, then the timer is deemed expired and the task is woken immediately and returned a successful completion status.

UNAPPROVED DRAFT. All rights reserved by VITA.
Do not specify or claim conformance to this document.

11.3. TIMER_EVENT_AFTER

Send event after a specified time interval.

Synopsis

```
timer_event_after( ticks, event, tmid )
```

Input Parameters

ticks	: integer	number of ticks to wait
event	: bit_field	event to send

Output Parameters

tmid	: timer_id	kernel defined timer identifier

Completion Status

OK	timer_event_after operation successful
INVALID_PARAMETER	a parameter refers to an illegal address
TOO_MANY_TIMERS	too many timers on the node

Description

This operation starts an event timer which will send the given events to the calling task after the specified number of ticks. The kernel returns an identifier which can be used to cancel the timer. If the node clock is set using the clock_set operation during this interval, the number of ticks left does not change.

UNAPPROVED DRAFT. All rights reserved by VITA.
Do not specify or claim conformance to this document.

11.4. TIMER_EVENT_WHEN

Send event at the specified wall time and date.

Synopsis

```
timer_event_when( clock, event, tmid )
```

Input Parameters

clock	: clock_buf	time and date to send event
event	: bit_field	event(s) to send

Output Parameters

tmid	: timer_id	kernel defined timer identifier

Completion Status

OK	timer_event_when operation successful
INVALID_PARAMETER	a parameter refers to an illegal address
INVALID_CLOCK	invalid clock value
TOO_MANY_TIMERS	too many timers on node

Description

This operation starts an event timer which will send the given events to the calling task at the given date and time. The kernel returns an identifier which can be used to cancel the timer.

If the node clock is set while the timer is running, the wall time at which the task is woken remains valid. If the node time is set to after the timer wake time, then the timer is deemed expired and the events are sent to the calling task immediately .

UNAPPROVED DRAFT. All rights reserved by VITA.
Do not specify or claim conformance to this document.

11.5. TIMER_CANCEL

Cancel a running event timer.

Synopsis

```
timer_cancel( tmid )
```

Input Parameters

tmid	: timer_id	kernel defined timer identifier

Output Parameters

<none>

Completion Status

OK	timer_cancel operation successful
INVALID_PARAMETER	a parameter refers to an illegal address
INVALID_ID	timer does not exist

Description

This operation cancels an event timer previously started using the timer_event_after or timer_event_when operations. The user specifies the timer using the identifier returned by these operations. If the given timer has expired or has been cancelled, the INVALID_ID completion status is returned.

UNAPPROVED DRAFT. All rights reserved by VITA.
Do not specify or claim conformance to this document.

12. INTERRUPTS

ORKID defines two operations which bracket interrupt handler code. It is up to each implementor to decide what functionality, to put in these operations.

Observation:

The kernel may use int_enter and int_exit with an Interrupt Service Routine code or task code is being executed. Typically int_exit will be used to decide if a scheduling action must take place in pre-emptive kernels.

UNAPPROVED DRAFT. All rights reserved by VITA.
Do not specify or claim conformance to this document.