10.1. CLOCK_SET

Set node time and date.

Synopsis

```
clock_set( clock )
```

Input Parameters

clock	: clock_buf	current time and date

Output Parameters

<none>

Completion Status

OK	clock_set 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 sets the node clock to the specified value. The kernel checks the supplied date and time in clock_buf to ensure that they are legal. This is purely a syntactic check - the operation will accept any legal value. The exact structure of the data supplied is language binding dependent.

10.2. CLOCK_GET

Get node time and date.

Synopsis

```
clock_get( clock )
```

Input Parameters

```
<none>
```

Output Parameters

```
clock       : clock_buf    current time and date
```

Completion Status

```
OK                         clock_get operation successful
INVALID_PARAMETER          a parameter refers to an illegal address
CLOCK_NOT_SET              clock has not been initialized
```

Description

This operation returns the current date and time in the node clock. If the node clock has not yet been set, then the CLOCK_NOT_SET completion status is returned. The exact structure of the clock_buf data returned is language binding dependent.

10.3. CLOCK_TICK

Announce a tick to the clock.

Synopsis

```
clock_tick(  )
```

Input Parameters

<none>

Output Parameters

<none>

Completion Status

OK	clock_tick operation successful

Description

This operation increments the current node time by one tick. There are no parameters and the operation always succeeds. Every node must contain a mechanism which keeps the node clock up to date by calling upon CLOCK_TICK.

11. TIMERS

ORKID defines two types of timers. The first type is the sleep timer. This type allows a task to sleep either for a given period, or up until a given time, and then wake and continue. Obviously a task can set only one such timer in operation at a time, and once set, it cannot be cancelled. These timers have no identifier.

The second type of timer is the event timer. This type allows a task to send events to itself either after a given period or at a given time. A task can have more than one event timer running at a time. Each event timer is assigned an identifier by the kernel when the event is set. This identifier can be used to cancel the timer.

Timers are purely local objects. They affect only the calling task, either by putting it to sleep or sending it events. Timers exist only while they are running. When they expire or are cancelled, they are deleted from the kernel data structure.

11.1. TIMER_WAKE_AFTER

Wake after a specified time interval.

Synopsis

```
timer_wake_after( ticks )
```

Input Parameters

ticks	: integer	number of ticks to wait

Output Parameters

<none>

Completion Status

OK	timer_wake_after operation successful
ILLEGAL_USE	operation not callable from XSR or ISR
INVALID_PARAMETER	a parameter refers to an illegal address

Description

This operation causes the calling task to be blocked for the given number of ticks. The task is woken after this interval has expired, and is returned a successful completion status. If the node clock is set using the clock_set operation during this interval, the number of ticks left does not change.