***Open Real-time Kernel Interface Definition Page 75 Draft 2.1***

**11. CLOCK**

Each ORKID kernel maintains a node clock. This is a single data object in the kernel data structure which contains the current date and time. The clock is updated at every tick, the frequency of which is node dependent. The range of dates the clock is allowed to take is implementation dependent.

In a multi—node system, the different node clocks will very likely be synchronized, although this is not necessarily done automatically by the kernel. Since nodes could be in different time zones in widely distributed systems, the node clock specifies the local time zone, so that all nodes can synchronize their clocks to the same absolute time.

The data structure containing the clock value passed in clock operations is language binding dependent. It identifies the date

and time down to the nearest tick, along with the local time zone. The time zone value is defined as the number of hours ahead (positive value) or behind (negative value) Greenwich Mean Time (GMT).

When the system starts up, the clock may be uninitialised. If this is the case, attempts at reading it before it has been set result in an error completion status, rather than returning a random value.

***Open Real-time Kernel Interface Definition Page 76 Draft 2.1***

11.1. CLOCK\_SET

**Set node time and date.**

Synopsis

clock\_set( clock )

Input Parameters

clock : clock\_buff current time and date

Output Parameters

<none>

Completion Status

OK clock\_set successful

ILLEGAL\_USE clock\_set not callable from ISR

INVALID\_PARAMETER a parameter refers to an invalid 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\_buff 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.

***Open Real-time Kernel Interface Definition Page 77 Draft 2.1***

**11.2. CLOCK\_GET**

Get node time and date.

Synopsis

c1ock\_get( clock )

Input Parameters

<none>

Output Parameters

clock : clock\_buff current time and date

Completion Status

OK c1ock\_get successful

INVALID\_PARAMETER a parameter refers invalid 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 and the contents of clock are undetermined. The exact structure of the c1ock\_buff data returned is language binding dependent.

***Open Real—time Kernel Interface Definition Page 78***

**11.3. CLOCK\_TlCK**

Announce a tick to the clock.

Synopsis

c1ock\_tick( )

**Input Parameters**

**<none>**

Output Parameters

<none>

**Completion Status**

OK lock\_tick successful

Description

This operation increments the current node time by one tick. There

are no parameters and the operation always succeeds. Nevertheless, the operation can be meaningless if the clock was not initialized beforehand. Every node must contain a mechanism which keeps the node clock up to date by calling upon clock\_tick.

***Open Real-time Kernel Interface Definition Page 79 Draft 2.1***

12. 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.

***Open Real-time Kernel Interface Definition Page 80 Draft 2.1***

12.1 . 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 successful

ILLEGAL\_USE timer\_wake\_after not callable from ISR

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.

***Open Real—time Kernel Interface Definition Page 81 Draft 2.1***

12.2. TlMER\_WAKE\_WHEN

Wake at a specified wall time and date.

**Synopsis**

timer\_wake\_when( clock )

Input Parameters

clock : c1ock\_buff time and date to wake

Output Parameters

<none>

**Completion Status**

OK timer\_wake\_when successful

ILLEGAL\_USE timer\_wake\_when not callable from ISR

INVALID\_PARAMETER a parameter refers to an invalid address

INVALID\_CLOCK invalid clock value

CLOCK\_NOT\_SET clock has not been initialized

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.

***Open Real-time Kernel Interface Definition Page 82 Draft 2.1***

**12.3. 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 successful ILLEGAL\_USE timer\_event\_after not callable from ISR INVALID\_PARAMETER a parameter refers to an invalid address TOO\_MANY\_OBJECTS 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.

***Open Real-time Kernel Interface Definition Page 83 Draft 2.1***

12.4. TlMER\_EVENT\_WHEN

Send event at the specified wall time and date.

Synopsis

timer\_event\_when( clock, event, tmid )

Input Parameters

clock : clock buff 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 successful

ILLEGAL\_USE timer\_event\_when not callable from ISR

INVALID\_PARAMETER A parameter refers to an invalid address

INVALID:CLOCK invalid clock value

TOO MANY OBJECTS too many timers on the node

CLOEK\_NOT\_SET clock has not been initialized

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 event(s) are sent remains valid. If the node time is set to after the value specified in the clock parameter, then the timer is deemed expired and the events are sent to the calling task immediately.

***Open Real-time Kernel Interface Definition Page 84 Draft 2.1***

12.5. TIMER\_EVENT\_EVERY

Send periodic event.

Synopsis

timer\_event\_every( ticks, event, tmid )

**Input Parameters**

ticks : integer number of ticks to wait between events

event : bit\_field event to send

Output Parameters

tmid : timer\_id kernel defined timer identifier

Completion Status

OK timer\_event\_every successful

ILLEGAL\_USE timer event\_every not callable from ISR INVALID\_PARAMETER a parameter refers to an invalid address TOO\_MANY\_OBJECTS too many timers on the node

Description

This operation starts an event timer which will periodically send the given events to the calling task with the periodicity specified by the 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 the life time of the timer, the number of ticks left until the next event does not change.

**Observation:**

***This provides a drift—free mechanism for sending an event at periodic intervals.***