4.1. TASK_CREATE

Create a task.

Synopsis

```
task_create( name, priority, stack_size, mode, options, tid )
```

Input parameters

name	: string	user defined task name
priority	: integer	initial task priority
stack_size	: integer	size in bytes of task's stack
mode	: bit_field	initial task mode
options	: bit_field	creation options

Output Parameters

tid	: task_id	kernel defined task identifier

Literal Values

mode	+ NOXSR	XSRs cannot be activated
	+ NOTERMINATION	task cannot be restarted or deleted
	+ NOPREEMPT	task cannot be preempted
	+ NOINTERRUPT	task cannot be interrupted
	= ZERO	no mode parameter set
options	+ GLOBAL	the new task will be visible throughout the system

Completion Status

OK	task_create successful
ILLEGAL_USE	task_create not callable from ISR
INVALID_PARAMETER	a parameter refers to an invalid address
INVALID_PRIORITY	invalid priority value
INVALID_MODE	invalid mode value
INVALID_OPTIONS	invalid options value
TOO_MANY_OBJECTS	too many tasks on the node or in the system
NO_MORE_MEMORY	not enough memory to allocate task data structure or task stack

Description

The task_create operation creates a new task in the kernel data structure. Tasks are always created in the node in which the call to task_create was made. The new task does not start executing code -this is achieved with a call to the task_start operation. The tid returned by the kernel is used in all subsequent **ORKID** operations (except task_ident) to identify the newly created task. If GLOBAL is specified in the options parameter, then the tid can be used anywhere in the system to identify the task, otherwise it can be used only in the node in which the task was created.

4.2. TASK_DELETE

Delete a task.

Synopsis

```
   task_delete( tid )
```

Input Parameters

tid	: task_id	kernel defined task identifier

Output Parameters

<none>

Literal Values

tid	= SELF	the calling task requests its own deletion

Completion Status

OK	task_delete successful
ILLEGAL_USE	task_delete not callable from ISR
INVALID_PARAMETER	a parameter refers to an invalid address
INVALID_ID	task does not exist
OBJECT_DELETED	originally existing task has been deleted before operation
OBJECT_NOT_LOCAL	task_delete not allowed on non-local task
OBJECT_PROTECTED	task in NOTERMINATION mode

Description

This operation stops the task identified by the tid parameter and deletes it from its node's kernel data structure. If the task's active mode has the parameter NOTERMINATION set, then the task will not be deleted and the completion status OBJECT_PROTECTED will be returned.

Observation:

The task_delete operation deallocates the task's stack but otherwise performs no 'clean-up' of the resources allocated to the task. It is therefore the responsibility of the calling task to ensure that all segments, buffers, etc., allocated to the task to be deleted have been returned.

For situations where one task wants to delete another, the recommended procedure is to ask this task to delete itself, typically using exceptions, or task_restart with a specific argument. In this way the task can release all its resources before deleting itself.

4.3 TASK_IDENT

Obtain the identifier of a task on a given node with a given name.

Synopsis

```
   task_ident( name, nid, tid )
```

Input Parameters

name	: string	user defined task name
nid	: node_id	node identifier

Output Parameters

tid	: task_id	kernel defined task identifier

Literal Values

nid	= LOCAL_NODE	the node containing the calling task
	= OTHER_NODES	all nodes in the system except the local node
	= ALL_NODES	all nodes in the system
name	= WHO_AM_I	returns tid of calling task

Completion Status

OK	task_ident successful
ILLEGAL_USE	task_ident not callable from ISR
INVALID_PARAMETER	a parameter refers to an invalid address
INVALID_ID	node does not exist
NAME_NOT_FOUND	task name does not exist on node
NODE_NOT_REACHABLE	node on which task resides is not reachable

Description

This operation searches the kernel data structure in the node(s) specified by nid for a task with the given name. If OTHER_NODES or ALL_NODES is specified, the node search order is implementation dependent. If there is more than one task with the same name in the node(s) specified, then the tid of the first one found is returned.

4.4. TASK_START

Start a task.

Synopsis

```
task_start( tid, start_addr, arguments )
```

Input Parameters

tid	: task_id	kernel defined task identifier
start_addr	: *	task start address
arguments	: *	arguments passed to task

Output Parameters

<none>

Completion Status

OK	task_start successful
ILLEGAL_USE	task_start not callable from ISR
INVALID_PARAMETER	a parameter refers to an invalid address
INVALID_ID	task does not exist
OBJECT_DELETED	originally existing task has been deleted before operation
INVALID_ARGUMENTS	invalid number or type or size of arguments
TASK_ALREADY_STARTED	task has been started already
NODE_NOT_REACHABLE	node on which task resides is not reachable

Description

The task_start operation starts a task at the given address. The task must have been previously created with the task_create operation.

* The specifications of start address and the number and type of arguments are language binding dependent.

4.5. TASK_RESTART

Restart a task.

Synopsis

```
   task_restart( tid, arguments )
```

Input Parameters

tid	: task_id	kernel defined identifier
arguments	: *	arguments passed to task

Output Parameters

<none>

Literal Values

tid	= SELF	the calling task restarts itself.

Completion Status

OK	task_restart successful
ILLEGAL_USE	task_restart not callable from ISR
INVALID_PARAMETER	a parameter refers to an invalid address
INVALID_ID	task does not exist
OBJECT_DELETED	originally existing task has been deleted before operation
INVALID_ARGUMENTS	invalid number or type or size of arguments
TASK_NOT_STARTED	task has not yet been started
OBJECT_PROTECTED	task in NOTERMINATION mode
NODE_NOT_REACHABLE	node on which task resides is not reachable

Description

The task_restart operation interrupts the current thread of execution of the specified task and forces the task to restart at the address given in the task_start call which originally started the task. The stack pointer is reset to its original value. No assumption can be made about the original content of the stack at this time. The task restarts executing with the priority and mode specified at task_create. All event and exception latches are clared and no XSRs are defined.

Any resources allocated to the task are not affected during the task_restart operation. The tasks themselves are responsible for the proper management of such resources through task_restart.

If the task's active mode has the parameter NOTERMINATION set, then the task will not be restarted and the completion status OBJECT_PROTECTED will be returned.

* The specification of the number and type of the arguments is language binding dependent.