GetEvent(*task_id*, *event_mask_ref*)

Get a copy of the event mask of task *task_id*. *event_mask_ref* is a pointer to an `EventMaskType` variable where the copy is written. Returns

E_OK	success
E_OS_ID†	task *task_id* does not exist
E_OS_ACCESS†	task *task_id* is not an extended task
E_OS_STATE†	task *task_id* is in `SUSPENDED` state

WaitEvent(*event_mask*) ⋈

If the none of the events in *event_mask* is set in the event mask of the caller, the caller is put in the `WAITING` state. Returns:

E_OK	success
E_OS_ACCESS†	the caller isn't an extended task
E_OS_RESOURCE†	the caller hold a resource
E_OS_CALLEVEL†	the caller is not a task

Resources

DeclareResource(*rez_id*)

Declare the existence of a resource so that it can be referenced.

GetResource(*rez_id*)

Get resource *rez_id*. The priority of the caller is raised to the priority of the resource if higher. Returns:

E_OK	success
E_OS_ID†	resource *rez_id* does not exist
E_OS_ACCESS†	the caller try to get a resource already held

ReleaseResource(*rez_id*) ⋈

Release resource *rez_id*. The priority of the caller returns to the priority it had before. Returns:

E_OK	success
E_OS_ID†	resource *rez_id* does not exist
E_OS_NOFUNC†	the caller try to release a resource it does not hold
E_OS_ACCESS†	the caller try to release a resource with a priority lower than the caller one

Messages

DeclareMessage(*mess_id*)

Declare the existence of a message so that it can be referenced.

SendMessage(*mess_id*, *data_ref*) ⋈

Send message *mess_id*. *data_ref* is a pointer to a variable containing the data to send. Returns:

E_OK	success
E_COM_ID†	message *mess_id* does not exist or has the wrong type

SendZeroMessage(*mess_id*) ⋈

Send signalization message *mess_id*. Returns:

E_OK	success
E_COM_ID†	message *mess_id* does not exist or has the wrong type

ReceiveMessage(*mess_id*, *data_ref*)

Receive message *mess_id*. *data_ref* is a pointer to a variable where the data are copied.

E_OK	success
E_COM_ID†	message *mess_id* does not exist or has the wrong type
E_COM_NOMSG	message *mess_id* is queued and the queue is empty
E_COM_LIMIT	message *mess_id* is queued and the queue has overflown

GetMessageStatus(*mess_id*)

Returns the status of a message:

E_COM_ID†	message *mess_id* does not exist
E_COM_NOMSG	message *mess_id* is queued and the queue is empty
E_COM_LIMIT	message *mess_id* is queued and the queue has overflown
E_OK	none of the above

OSEK QRDC

Jean-Luc Béchennec – LS2N

v1.0 – September 2018

Data types

`StatusType`	error code returned by a service
`AppModeType`	an application mode
`TaskType`	identifier of a task
`TaskStateType`	state of a task (`SUSPENDED`, `READY`, `RUNNING` or `WAITING`)
`AlarmType`	identifier of an alarm
`AlarmBaseType`	counter attributes
`TickType`	number of ticks
`EventMaskType`	a set of events
`ResourceType`	identifier of a resource
`MessageType`	identifier of a message

Services

Each service returns an error code except `GetActiveApplicationMode`. If the OS has been compiled in EXTENDED configuration additional error codes may be returned and are suffixed by a †. Services suffixed by a ⋈ lead to a rescheduling.

Operating system

StartOS(*app_mode*)

Start the operating system in application mode *app_mode*. Does not return.

ShutdownOS(*error*)

Shutdown the operating system with error code *error*. Does not return.

GetActiveApplicationMode()

Returns the application mode used to start the operating system.

Tasks

DeclareTask(*task_id*)

Declare the existence of a task so that it can be forward referenced.

ActivateTask(*task_id*) ⋈

Activate task *task_id*. If task *task_id* has a priority greater than the caller priority, the caller is preempted. Returns:

E_OK	success
E_OS_LIMIT	too many activation of *task_id*
E_OS_ID†	task *task_id* does not exist

TerminateTask() ⋈

Terminate the caller. Returns:

E_OK	success
E_OS_RESOURCE†	the caller hold a resource
E_OS_CALLEVEL†	the caller is not a task

ChainTask(*task_id*) ⋈

Terminate the caller and activate *task_id*. Returns:

E_OK	success
E_OS_LIMIT	too many activation of *task_id*
E_OS_ID†	task *task_id* does not exist
E_OS_RESOURCE†	the caller hold a resource
E_OS_CALLEVEL†	the caller is not a task

Schedule() ⋈

Call the scheduler. Returns:

E_OK	success
E_OS_RESOURCE†	the caller hold a resource
E_OS_CALLEVEL†	the caller is not a task

GetTaskID(*task_id_ref*)

Get the task identifier of the task which is currently running. *task_id_ref* is a pointer to a `TaskType` variable where the task identifier of the running task is written. Returns:

E_OK	success

GetTaskState(*task_id*, *task_state_ref*)

Get the task state of task *task_id*. *task_state_ref* is a pointer to a `TaskState` variable where the state is written. Returns:

E_OK	success
E_OS_ID†	task *task_id* does not exist

Alarms

DeclareAlarm(*alarm_id*)

Declare the existence of an alarm so that it can be referenced.

GetAlarm(*alarm_id*, *tick_ref*)

Get the remaining tick count of alarm *alarm_id* before the alarm reaches the date. *tick_ref* is a pointer to a `TickType` variable where the remaining tick count is written. Returns:

E_OK	success
E_OS_NOFUNC	alarm *alarm_id* is not started
E_OS_ID†	alarm *alarm_id* does not exist

GetAlarmBase(*alarm_id*, *info_ref*)

Get the information about the underlying counter of alarm *alarm_id*. *info_ref* is a pointer to a `AlarmBaseType` variable where the information is written. A `AlarmBaseType` is a `struct` with 3 fields: `maxallowedvalue`, `ticksperbase` and `mincycle`. Returns:

E_OK	success
E_OS_ID†	alarm *alarm_id* does not exist

SetRelAlarm(*alarm_id*, *offset*, *cycle*)

Start alarm *alarm_id*. After *offset* ticks the alarm expire and its action is executed. *offset* shall be > 0. If *cycle* is > 0 the alarm is restarted and expire every *cycle* ticks. Both *offset* and *cycle* shall $\in$ [`MINCYCLE`, `MAXALLOWEDVALUE`]. Returns:

E_OK	success
E_OS_NOFUNC	alarm *alarm_id* is already started
E_OS_ID†	alarm *alarm_id* does not exist
E_OS_VALUE†	*offset* and/or *cycle* out of bounds

SetAbsAlarm(*alarm_id*, *date*, *cycle*)

Start alarm *alarm_id*. At next counter *date* the alarm expire and its action is executed. If *cycle* is > 0 the alarm is restarted and expire every *cycle* ticks. *date* shall be $\leq$ `MAXALLOWEDVALUE`. *offset* shall $\in$ [`MINCYCLE`, `MAXALLOWEDVALUE`]. Returns:

E_OK	success
E_OS_NOFUNC	alarm *alarm_id* is already started
E_OS_ID†	alarm *alarm_id* does not exist
E_OS_VALUE†	*date* and/or *cycle* out of bounds

CancelAlarm(*alarm_id*)

Stop alarm *alarm_id*. Returns:

E_OK	success
E_OS_NOFUNC	alarm *alarm_id* is not started
E_OS_ID†	alarm *alarm_id* does not exist

Events

DeclareEvent(*event_id*)

Declare the existence of an event so that it can be referenced.

SetEvent(*task_id*, *event_mask*) ⋈

Set event(s) *event_mask* to task *task_id*. If task *task_id* was waiting for one of the events of *event_mask* and it has a higher priority than the caller, the caller is preempted. Returns:

E_OK	success
E_OS_ID†	task *task_id* does not exist
E_OS_ACCESS†	task *task_id* is not an extended task
E_OS_STATE†	task *task_id* is in `SUSPENDED` state

ClearEvent(*event_mask*)

Clear the event(s) of the caller according to events set in *event_mask*. Returns:

E_OK	success
E_OS_ACCESS†	the caller is not an extended task
E_OS_CALLEVEL†	the caller is not a task